Studies on Reliability Analysis and Design of Complex Systems(Dissertation_全文)

AUTHOR(S):

Nakashima, Kyoichi

STUDIES ON
RELIABILITY ANALYSIS AND DESIGN
OF COMPLEX SYSTEMS

KYOICHI NAKASHIMA

MARCH 1980

STUDIES ON
RELIABILITY ANALYSIS AND DESIGN
OF COMPLEX SYSTEMS

KYOICHI NAKASHIMA

Submitted in partial satisfaction of the
requirements of the degree
DOCTOR OF ENGINEERING
at
KYOTO UNIVERSITY
Kyoto, JAPAN

MARCH 1980

Preface

The need for high reliable system and reliability analysis became keenly felt after the Second World War. Such need has become greater with the development of large systems, such as power plants, chemical plants, aircraft systems, computer systems and communication systems. The reliability of system has hence been a topic that interests many engineers and researchers.

This thesis is devoted to extending the current state of reliability analysis and design techniques of complex systems. The particular emphasis is put on techniques for probabilistic evaluations using fault trees & networks, and their algorithms. Such techniques would be useful for assuring system designers of estimating system reliability and making their decisions with high rapidity and precision.

Chapter 1 proves some properties of coherent structure function for logic trees containing mutually exclusive primary events. These properties are useful for the systematic reliability analysis of the system that the failures of components are mutually dependent.

Chapter 2 through Chapter 4 present efficient algorithms for evaluating fault trees. The fault tree technique is a powerful tool for system reliability/safety analysis and has interested analysts of complex systems lately.

Chapter 2 presents a representation of fault tree suitable

for the computer processing. Reverse Polish sequence is extended for fault tree containing k-out-of-n logic gates, and tree sequence is newly introduced. Some useful properties and algorithms concerning tree sequence are given. Chapter 3 presents an efficient bottom-up algorithm for enumerating all minimal cut sets of fault tree. Chapter 4 presents a method for computing the top event probability of fault tree through the combined use of reverse Polish sequence and tree sequence. The method can be applied to the sensitivity analysis. The computational results obtained by applying the method to the containment spray injection system of PWR nuclear power plant are shown.

Chapter 5 is devoted to develop the method for the reliability analysis of network. The method is based on the strategy of first obtaining the transmission Boolean function of network and then executing reliability calculation. Our effort is focused on representing the transmission Boolean function as shortly as possible.

How to make the tradeoffs between reliability and cost is one of important problems to a system designer. Chapter 6 formulates the problems of optimizing both of two ways of improving system reliability, i.e., adding redundant components and increasing component reliability, under time-dependent reliability, and presents a solution method for solving them.

Acknowledgement

I am truly indebted to many individuals and institutions for making this thesis possible.

I would like to acknowledge, first of all, Professor Y. Hattori of Kyoto University for his constant guidance, encouragement and suggestions during every phase of this work.

I wish to thank Professor K. Yamato of Himeji Institute of Technology for his continued advice and encouragement. I am also grateful to Professor K. Kawanishi of Tottori University who aroused my interests in the field of reliability when he was a professor of Himeji Institute of Technology.

I would like to thank Professor H. Mine of Kyoto University and Dr. R. A. Evans, Editor of IEEE Transactions on Reliability, for their useful comments and suggestions.

I am indebted to Dr. Y. Nakagawa for his continued help and valuable discussions about reliability optimization.

My sincere gratitude goes to the members of Professor Hattori's Laboratory, Kyoto University, for their helpful discussions, and the members of Department of Electronics, Himeji Institute of Technology, for having given me the opportunity of research study at Kyoto University. I am also grateful to several graduates of Himeji Institute of Technology, who helped me in getting on with this work when they studied at my research laboratory.

Most importantly, I wish to thank my mother and father for all

the years of their patient support, and for their insuring the completion of my college education.

Finally, I give my deep thanks to my wife for her constant help and encouragement.

Contents

Introduction

The development of modern engineering has enabled us to assemble large systems. However, as the scale of system becomes large, the possibility that the failure of system occurs increases and its occurrence often exerts bad influences economically and socially, moreover occasionally endanger people's lives. Therefore, for a complex system that has the large number of components, it is required to develop the techniques for estimating the reliability and safety of system as precisely as possible and designing the systems with high reliability and safety.

When we would analyze the reliability of complex system, it is necessary to express the system in a form suitable for the analysis. As the means of visually expressing the functional relation between the system and its components, we often use diagrams, such as a reliability block diagram with series-parallel or non-series-parallel structure, a network with directed or undirected branches etc. As the logical expressions relating the failure or success of system to the failures or successes of its components, we have diagramatically a logic tree [12] which is classified into either fault tree for system failure or success tree for system success,

an event tree [68], a cause-consequence chart [57], etc., and have algebrarically the structure function that is a function relating the state variable of system to the state variables of components. The reliability analysis of complex system ia thus executed through the process of first obtaining an appropriate visual or logical expression of system and then estimating system reliability by using those expressions. Many efforts have been made to develop efficient techniques for executing the above process, and will have to be made in future as well. This thesis is provided to extend the current state of such techniques.

Various studies on the structure function of complex system have been made since the paper by Moore and Shannon [39] was published in 1956. The stucture function that has monomtonicity in state variables of components was introduced by Mine [36] and afterwards Birnbaum et al. [9] called it a coherent structure function. The main results of former studies on the coherent structure function have been reviewed by Barlow and Proschan [3, 6]. In former studies, the states of each component are restricted to two states of operating state and failed state. In this case, the coherent structure function contains only independent binary variables. However, the structure function might contain mutually dependent binary variables, as seen in the case where mutually exclusive primary events (e.g., multimodal failures of a component, such as open and closed failures of a switch or valve) appear in a logic tree. Chapter 1 extends the definitions of coherent structure function and other terms for logic trees containing mutually exclusive primary events, and presents some properties of coherent structure function and their applications. These results will enable us to do the systematic reliability analysis of the system that the failures of components are mutually dependent [48, 53].

In Chapter 2 through Chapter 4, efficient algorithms for analyzing fault trees are presented [47, 49, 51, 54].

Recently fault trees are often used for the reliability analysis and design of systems, such as electrical, airplane, nuclear reactor and chemical plant systems [23, 56, 68, 59], of which high safety or reliability is required. This is because, in such system, all events that cause a specific undesired system failure must be investigated and an effective counterplan which prevents the occurrence of system failure should be worked out.

The analysis of fault tree is roughly divided into the stage of constructing the fault tree and the stage of obtaining qualitative or quantitative information about system failure from the constructed fault tree. Generally speaking, manual construction of fault tree is much laborious and time-consuming, moreover is liable to oversight and omission [19]. To get out of these difficulties, a few computer-aided construction techniques [18, 59, 35] have been presented. Referring to the latter stage, we can obtain information about system failure, such as minimal cut sets, the probability or frequency of occurrence of the top event, sensitivity, the importance of each primary event, etc., from the fault tree. However, according as the scale of fault tree becomes large, the computation time and storage requirement for obtaining such information extremely increase even if a digital computer is used: Therefore, efficient algorithms are required for the large scale fault tree. Chapters 2, 3, 4 are put to respond to this requirement.

The first step of fault tree evaluation is to obtain a representation of fault tree suitable for computer processing. There are the existing methods [8, 32] which apply list processing technique using reverse Polish sequence (expression). In Chapter 2, the reverse Polish sequence is extendedly defined for the fault tree containing k-out-of-n gates, and a tree sequence composed of only integer values is newly introduced to implement effectively the processing by using the recursive character of tree: the tree sequence represents the branching structure of tree completely.

Some properties and algorithms concerning the tree sequence are indicated. By using those algorithms, we can process the tree simply through the algebric operation of the values composing the tree sequence.

A basic characteristic of fault tree is a minimal cut set that is a minimal set of primary events which must all occur for the top event to occur. The list of all minimal cut sets is useful for various probabilistic evaluations of fault tree, the determination of critical path to the occurrence of system failure, etc. [5, 20, 69] Chapter 3 presents an efficient algorithm for enumerating all minimal cut sets of fault tree. This algorithm improves the conventional bottom-up algorithm [8] so as to obtain all minimal cut sets more quickly. The improvement is to reduce the number of checks of redundant terms for the logical product of two reduced sum-of-product forms. The computational results for several examples are given to demonstrate the efficiency of this algorithm.

The probabilistic evaluation of fault tree (e.g., top event probability, unavailability, failure intensity) is useful for the estimation of the possibility of occurrence of system failure or accident, the calculation of the importance of primary event to top event which enables us to find the optimal way of improving system reliability, etc. A major goal of probabilistic evaluation is to compute the probability of occurrence of top event (top event probability). Formerly the main efforts have been exerted to generate minimal cut sets and then to compute the top event probability by applying the addition rule in probability theory or by generating an equivalent disjoint sum-of-product form. However, such techniques are so time-consuming for the large scale fault tree that we oblige to settle for approximate results.

The method presented in Chapter 4 computes the exact value of top event probability without going through the process of generating minimal cut sets. The feature of the method is to

repeat recursively the partition and reduction of fault trees and to reduce to the computations of top event probability of simple fault trees that contain no repeated primary events. In the computer processing, the reverse Polish sequence and tree sequence presented in Chapter 2 are used. The method can be applied to the analysis of fault tree containing mutually exclusive primary events and the sensitivity analysis. The results obtained by applying the method to the containment spray injection system of PWR nuclear power plant [68] are shown.

Physical systems, such as communication networks, transportation networks, power transmission lines, chemical process systems etc., can be represented by networks with directed or undirected branches [2, 15, 25]. The reliability analysis of such systems has been the subject that system researchers are interested in. In the analysis, the network is treated as a probabilistic graph in which each branch or node has a probability that it is good (in the operating state) or bad (in the failed state).

A major problem of network reliability analysis is to calculate the node-pair reliability, i.e., the probability that there exists at least one path from the source node to the sink node. Most of existing methods are based on the strategy of calculating the node-pair reliability directly through state enumeration, path (cutset) enumeration, reduction, decomposition etc. The method presented in Chapter 5 is based on the strategy of first obtaining the transmission Boolean function of the network and then executing reliability calculation [52]. It is of significance to store the transmission Boolean function obtained, because it can widely be used for various cases, e.g., the case where nodes and branches are mutually dependent. In the method, the reverse Polish sequence and tree sequence of transmission Boolean function are obtained by a backtracking technique repeating recursively two processes, reduction of a network and decomposition of completely reduced network, until reaching series-parallel structure.

At the stage of designing a system, one of important problems that we are faced with is how to make tradeoffs between reliability and cost. We usually consider two ways of improving system reliability: 1) adding redundant components and 2) increasing reliability of a component. The latter way corresponds to tightening quality control in producing each component, developing new components with higher reliability, etc. Both ways usually requires the increase of system cost.

Many methods have been presented for optimizing system reliability under given constraints; Recently the reviews on such methods have been made by some authors [45, 65]. Most of them consider only either of two ways and assume time-independent reliability. Only a few researchers [38, 66] consider the problem of determining both optimal number of redundant components and optimal component reliability which is a mixed integer programming problem; They also assume time-independent reliability.

Chapter 6 formulates the problem of optimizing both of two ways of improving system reliability under time-dependent reliability. System reliability is monotonically decreasing with time. For this time-dependency, we adopt as the performance index the mission time that system reliability is above a preassigned value. The solution methods have been presented for the problems of maximizing the above mission time under system cost constraint and minimizing the system cost under mission time constraint.

Chapter 1

Some Properties of Logic Trees Containing Mutually Exclusive Primary Events

This chapter analyzes logic trees containing mutually exclusive primary events. A logic tree is similar to a fault tree, but can contain successes as well as failures. Coherent structure function and other basic terms are defined. Some useful properties of coherent structure function are presented which include, as special cases, the previous results for logic trees containing no mutually exclusive primary events. A method is provided for obtaining an effective upper bound to probability of occurrence of the top event for logic trees with noncoherent structure function.

1.1 Introduction

A logic tree is a logic diagram having a tree structure whose construction begins with a specific event (viz. top event) and continues by branching deductively until reaching causative events that cannot or need not be developed further (viz. primary event). Each branching point is represented by an appropriate logic gate. The top event can be represented as a Boolean structure function of primary events appearing in the logic gate.

Many discussions [6, 9, 14, 36] have been presented about coherent (or monotone) structure functions for logic trees containing no mutually exclusive primary events. However, mutually exclusive primary events might appear in a logic tree: e.g., multimodal failures of a unit, such as open and closed failures of a switch or valve. A logic tree might contain primary events which are statistically-dependent but not mutually exclusive. In this case the logic tree can be transformed into an equivalent logic tree containing mutually exclusive primary events as follows. Suppose that two primary events "unit 1 fails"(denoted E_1) and "unit 2 fails" (denoted E_2) are statistically-dependent. Let T_1, T_2, T_3 be the events, "unit 1 fails and unit 2 operates", "unit 1 operates and unit 2 fails" and "unit 1 fails and unit 2 fails", respectively. Then the logic tree can be transformed into an equivalent logic tree containing mutually exclusive primary events T_1, T_2, T_3 by replacing E_1 and E_2 with $T_1 \vee T_3$ and $T_2 \vee T_3$, respectively. For logic trees containing three or more statistically-dependent primary events, equivalent logic trees can be obtained similarly.

When we analyze logic trees containing mutually exclusive primary events, how shall we extend terminology and methods that have been used for analyzing logic trees containing no mutually exclusive primary events ? This chapter answers this question and provides an approximate method for probabilistic evaluation. Section 1.2 extends definitions of coherent structure function and other terms. Section 1.3 presents some properties of coherent structure functions which are useful for analyzing logic trees. Section 1.4 explains the definitions and properties presented in Section 1.2 - 1.3 by three examples and illustrates a method for obtaining an effective upper bound to the probability of occurrence of the top event for a logic tree with noncohcrent structure function.

1.2 Definitions on Logic Trees Containing Mutually Exclusive Primary Events

Let T, E_{ij} be the top event and primary events, $i = 1, 2, \ldots, n$; $j = 1, 2, \ldots, \alpha_i$. T or E_{ij} is also used as binary variable having the value 1 if that event has occurred and 0 otherwise. Let $\Phi_i \equiv \{E_{i1}, E_{i2}, \ldots, E_{i\alpha_i}\}$ be the set of mutually exclusive and exhaustive primary events; $i = 1, 2, \ldots, n$. Then,

$$E_{ij}\, E_{ik} = 0, \qquad \text{for } j \neq k \text{ and } j, k = 1, 2, \ldots, \alpha_i,$$

$$E_{i1} \vee E_{i2} \vee \ldots \vee E_{i\alpha_i} = 1, \tag{1.1}$$

$$i = 1, 2, \ldots, n.$$

The left hand sides of (1.1) are logical product (AND) and logical sum (OR). Eq. (1.1) means that only one event of Φ_i always has the value 1 and all other events of Φ_i have 0. Hereinafter, the state that $E_{i\nu} = 1$ $(1 \leq \nu \leq \alpha_i)$ and $E_{ij} = 0$ for $j \neq \nu$ and $j = 1, 2, \ldots, \alpha_i$ is represented simply by $E_{i\nu} = 1$.

Define the vectors $\boldsymbol{E}_i \equiv (E_{i1}, E_{i2}, \ldots, E_{i\alpha_i})$, $i = 1, 2, \ldots, n$ and $\boldsymbol{E} \equiv (\boldsymbol{E}_1, \boldsymbol{E}_2, \ldots, \boldsymbol{E}_n)$. Then T is represented as the Boolean structure function of $\boldsymbol{E}$, i.e.,

$$T = \psi(\boldsymbol{E}). \tag{1.2}$$

Some primary events belonging to Φ_i might not appear in the fault tree: e.g., the primary event corresponding to the normally operating state of a unit would not cause system failure, and hence that primary event does not appear in the fault tree.

(D.1) Consider proper subset $\Omega_i \equiv \{E_{i(1)}, E_{i(2)}, \ldots, E_{i(\beta_i)}\}$ $(\beta_i < \alpha_i)$ of Φ_i, letting $\{(1), (2), \ldots, (\alpha_i)\}$ be a permutation of $\{1, 2, \ldots, \alpha_i\}$. Let $\boldsymbol{E}_i^* \equiv (E_{i(1)}, E_{i(2)}, \ldots, E_{i(\beta_i)})$ and

$E^* \equiv (E_1^*, E_2^*, \ldots, E_n^*)$. $\psi(E)$ is said to be *positive* in E^* if $\psi(E)$ can be represented as a sum-of-product (s.o.p.) form containing only primary events which belong to $\bigcup_{i=1}^{n} \Omega_i$. (Note that, in the s.o.p. form, there is not any term (logical product) that contains two or more mutually exclusive primary events in Ω_i together, from the relation of Eq. (1.1).)

(D.2) E_{ij} is *irrelevant* to $\psi(E)$ if $\psi(1_{ij}, E) = \psi(0_i, E)$ for all $(\bullet_i, E)$. Otherwise E_{ij} is *relevant* to $\psi(E)$. The notations $(1_{ij}, E)$, $(0_i, E)$, $(\bullet_i, E)$ are defined as follows:

$$(1_{ij}, E) \equiv (E_1, \ldots, E_{i-1}, 0, \ldots, E_{ij} = 1, 0, \ldots, 0, E_{i+1}, \ldots, E_n)$$

$$(0_i, E) \equiv (E_1, \ldots, E_{i-1}, E_i = 0, E_{i+1}, \ldots, E_n)$$

$$(\bullet_i, E) \equiv (E_1, \ldots, E_{i-1}, E_{i+1}, \ldots, E_n)$$

These definitions and notations are cited in [6].

(D.3) $\psi(E)$ is *coherent* if there exists a set $\{\Omega_1, \Omega_2, \ldots, \Omega_n\}$ such that $\psi(E)$ is positive in E^* and all primary events belonging to $\bigcup_{i=1}^{n} \Omega_i$ are relevent to $\psi(E)$. Then $\bigcup_{i=1}^{n} \Omega_i$ is said to be *essential set*.

$\psi(E)$ is *noncoherent* if all primary events belonging to one or more sets Ω_i are relevant to $\psi(E)$. Then those sets Ω_i are said to be *complete sets*.

By this definition, coherent structure function is extended for a logic tree containing mutually exclusive primary events. The special case of $\alpha_1 = \alpha_2 = \ldots = \alpha_n = 2$ and $\beta_1 = \beta_2 = \ldots = \beta_n = 1$ corresponds to a logic tree containing no mutually exclusive primary events.

(D.4) $\psi(\boldsymbol{E})$ is *monotone nondecreasing* in $\boldsymbol{E}^*$ if the relation $\psi(\boldsymbol{E}^{(1)}) \geq \psi(\boldsymbol{E}^{(2)})$ holds for all pairs of vectors $\boldsymbol{E}^{(1)}$, $\boldsymbol{E}^{(2)}$ in which the elements of $\boldsymbol{E}^*$ have the values of $\boldsymbol{E}^{*(1)}$, $\boldsymbol{E}^{*(2)}$, respectively, satisfying $\boldsymbol{E}^{*(1)} > \boldsymbol{E}^{*(2)}$ and other elements are identical.

(D.5) Let $\chi(\boldsymbol{E}^*)$ be a s.o.p. form of $\psi(\boldsymbol{E})$ which contains only primary events belonging to essential set $\bigcup_{i=1}^{n} \Omega_i$ when $\psi(\boldsymbol{E})$ is coherent. A *minimal cut set* is a set of primary events having the value 1 in a vector $\boldsymbol{E}^{*(c)}$ such that $\chi(\boldsymbol{E}^{*(c)}) = 1$ holds and $\chi(\boldsymbol{E}^*) = 0$ holds for all vectors $\boldsymbol{E}^*$ which satisfy $\boldsymbol{E}^* < \boldsymbol{E}^{*(c)}$. Then $\boldsymbol{E}^{*(c)}$ is a *minimal cut vector*. For a success tree the term *path* is generally used in place of the term *cut*.

(D.6) If $\psi(\boldsymbol{E})$ is coherent and the relation

$$S(T = 1 | E_{ij} = 1) \cap S(T = 0 | E_{ik} = 1) = \phi \tag{1.3}$$

holds for two elements E_{ij}, E_{ik} of essential set Ω_i, then $\psi(\boldsymbol{E})$ is *positive in the transition from* $E_{ij} = 1$ *to* $E_{ik} = 1$, denoted $E_{ij} \xrightarrow{p} E_{ik}$, where $S(A|B)$ means the set of states such that A holds, given the condition B, and ϕ is empty set.

1.3 Properties of Coherent Structure Function

A coherent structure function has the following properties. The proofs are given in Appendix A.

P1) The necessary and sufficient conditions for $\psi(\boldsymbol{E})$ to be positive in $\boldsymbol{E}^*$ are that a) $\psi(\boldsymbol{E})$ is monotone nondecreasing in $\boldsymbol{E}^*$ and b) (1.4) holds for $\Omega_i^c \equiv \{E_{i(\beta_i+1)}, E_{i(\beta_i+2)}, \cdots\cdots, E_{i(\alpha_i)}\}$, i.e., the complement of Ω_i in Φ_i.

$$\psi(1_{i(\beta_i+1)}, \boldsymbol{E}) = \psi(1_{i(\beta_i+2)}, \boldsymbol{E}) = \cdots\cdots = \psi(1_{i(\alpha_i)}, \boldsymbol{E}),$$

$$i = 1, 2, \ldots, n. \tag{1.4}$$

P2) Assume that $\chi(\boldsymbol{E}^*)$ does not contain two or more identical terms; this assumption means that, if a s.o.p. form of $\psi(\boldsymbol{E})$ has contained two or more identical terms, then they are preliminarily reduced to one term by the idempotence rule.

Let h be the number of terms in $\chi(\boldsymbol{E}^*)$ and Q_μ be the set of primary events appearing in term μ of $\chi(\boldsymbol{E}^*)$; $\mu = 1, 2, \ldots, h$. Then, all minimal cut sets of $\psi(\boldsymbol{E})$ are obtained by discarding from $\{Q_1, Q_2, \ldots, Q_h\}$ all Q_μ which satisfy $Q_\mu \supset Q_\nu$ for some Q_ν, $\nu \neq \mu$, $\nu = 1, 2, \ldots, h$.

P3) The following 3 relations hold if and only if $E_{ij} \xrightarrow{p} E_{ik}$.

1) $\psi(1_{ij}, \boldsymbol{E}) \leq \psi(1_{ik}, \boldsymbol{E})$ (or $\chi(1_{ij}, \boldsymbol{E}^*) \leq \chi(1_{ik}, \boldsymbol{E}^*)$),

$$\text{for all } (\bullet_i, \boldsymbol{E}). \tag{1.5}$$

2) $S(T = 0 | E_{ij} = 1) \cap S(T = 1 | E_{ik} = 1) = S(T = 1 | E_{ik} = 1)$

$$- S(T = 1 | E_{ij} = 1). \tag{1.6}$$

3) Let C be a minimal cut set containing E_{ij}. Then there exists at least one minimal cut set that is the union of $\{E_{ik}\}$ and a subset of $C - \{E_{ij}\}$.

Note: If and only if $E_{ij} \xrightarrow{p} E_{ik}$ and $E_{ik} \xrightarrow{p} E_{ij}$, then $S(T = 1 | E_{ij} = 1) = S(T = 1 | E_{ik} = 1)$ holds, and the set obtained by replacing E_{ij} (or E_{ik}) with E_{ik} (or E_{ij}) in a minimal cut set containing E_{ij} (or E_{ik}) is also a minimal cut set. The latter fact

means that we can reduce two primary events E_{ij}, E_{ik} to a primry event $E_{ij} \vee E_{ik}$.

P4) Suppose that elements (primary events) of a set among n sets Φ_i, $i = 1, 2, \ldots, n$ are statistically independent of elements of the other sets.

Partition the set of all minimal cut sets of $\psi(\boldsymbol{E})$ into sets $F_l = \{C_{l1}, C_{l2}, \ldots, C_{lt_l}\}$, $l = 1, 2, \ldots, s$ so that $\bigcup_{m=1}^{t_l} C_{lm}$ might not contain two or more primary events belonging to an essential set Ω_i. Then, an upper bound to the probability of occurrence of the top event is

$$\Pr\{T = 1\} \leq \sum_{l=1}^{s}[1 - \prod_{m=1}^{t_l}(1 - P_{lm})], \tag{1.7}$$

where P_{lm} is the product of probabilities of occurrence of primary events belonging to minimal cut set C_{lm}.

P1 shows that a coherent structure function keeps monotonicity by extending the definition as given in Section 1.2. P2 guarantees that all minimal cut sets of coherent structure function are obtained by applying the absorption rule to terms of $\chi(\boldsymbol{E}^*)$. P3 presents the important relations between two primary events belonging to an essential set. Eq. (1.6), suggested by Murchland [42], simplifies computing the probability that the occurrence of top event and the transition from state $E_{ij} = 1$ to $E_{ik} = 1$ coincide. P3-3 is useful for checking if '$E_{ij} \xrightarrow{p} E_{ik}$' is true, by using minimal cut sets. P4 is an effective upper bound to the probability of occurrence of the top event and extends the property [14], thus far known for a coherent structure function containing no mutually exclusive primary events.

1.4 Explanation by Examples

1.4.1 Examples of Coherent Structure

Example 1

Consider the system of 3 switches SW_1, SW_2, SW_3, connected as shown in Fig. 1. The logic tree is shown in Fig. 2. The top event and primary events are:

T system fails open or fails closed

E_{i1} SW_i normally operates

E_{i2} SW_i fails closed

E_{i3} SW_i fails open

$i = 1, 2, 3$

$\Phi_i = \{E_{i1}, E_{i2}, E_{i3}\}$, $i = 1, 2, 3$ and $\alpha_1 = \alpha_2 = \alpha_3 = 3$.

$$\psi(\boldsymbol{E}) = E_{12}(E_{22} \vee E_{32}) \vee (E_{13} \vee E_{23}E_{33}). \tag{1.8}$$

Unpack $\psi(\boldsymbol{E})$ by the distribution rule.

$$\chi(\boldsymbol{E}^*) = E_{12}E_{22} \vee E_{12}E_{32} \vee E_{13} \vee E_{23}E_{33}. \tag{1.9}$$

$\psi(\boldsymbol{E})$ is coherent since $\chi(\boldsymbol{E}^*)$ contains only primary events belonging to $\Omega_i = \{E_{i2}, E_{i3}\}$, $i = 1, 2, 3$.

Since each term of the right hand side of (1.9) is not absorbed by any other term, the set of primary events contained in each term is a minimal cut set. It is verified by P3-3 that $E_{12} \xrightarrow{p} E_{13}$ holds, but the positive relation does not hold between E_{22} and E_{23} nor between E_{32} and E_{33}.

Four minimal cut sets are partitioned into the following two sets F_1, F_2.

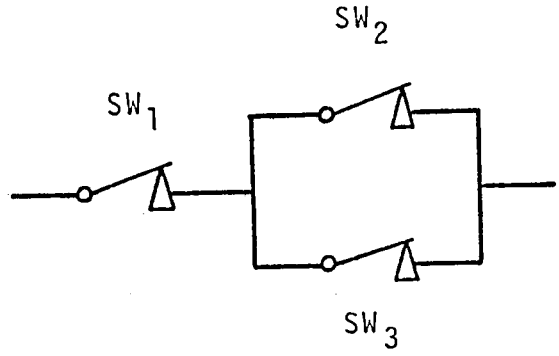


Fig. 1 Example 1 of System

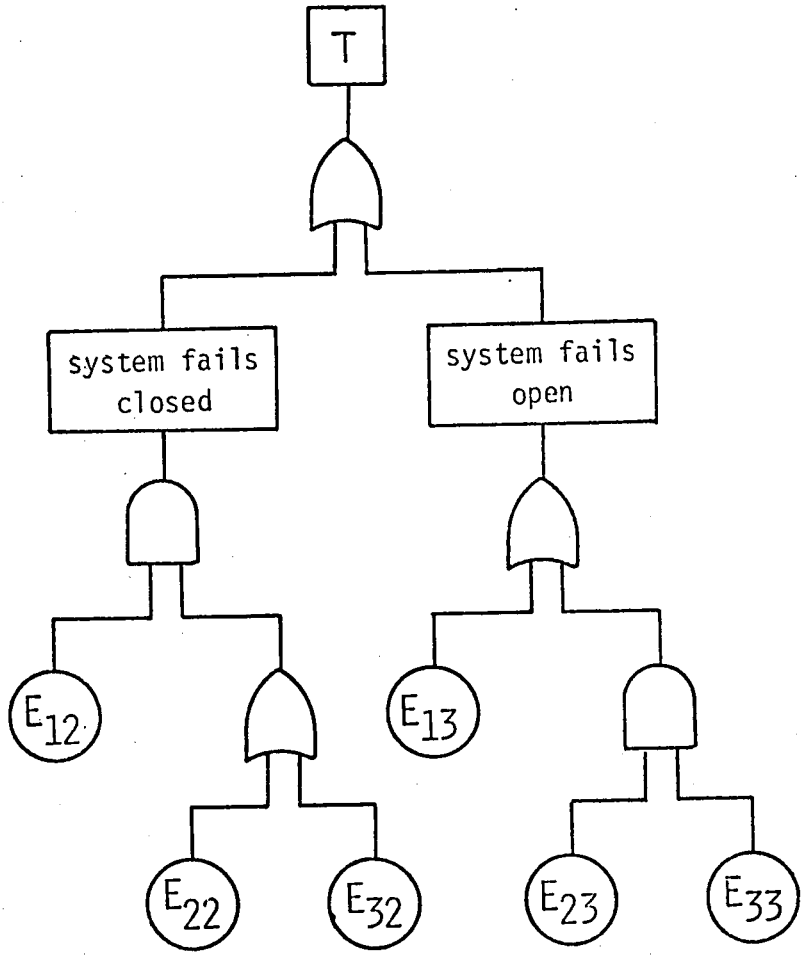


Fig. 2 Logic Tree for Example 1

$$F_1 = \{C_{11}, C_{12}\}, \quad F_2 = \{C_{21}, C_{22}\},$$

where $C_{11} = \{E_{12}, E_{22}\}$, $C_{12} = \{E_{12}, E_{32}\}$, $C_{21} = \{E_{13}\}$, $C_{22} = \{E_{23}, E_{33}\}$. Applying P4,

$$\Pr\{T = 1\} \leq 1 - (1 - p_{12}p_{22})(1 - p_{12}p_{32}) + 1 - (1 - p_{13}) \times (1 - p_{23}p_{33}), \tag{1.10}$$

where p_{ij} is the probability of occurrence of E_{ij}.

Example 2

Consider the communication network with 4 perfectly reliable nodes such as shown in Fig. 3. The logic tree for this network is shown in Fig. 4. The top event and primary events are:

T source node can communicate to sink node

E_{i1} branch i functions

E_{i2} branch i fails

$$i = 1, 2, \ldots, 6$$

The Boolean structure function for this logic tree is:

$$\psi(\boldsymbol{E}) = E_{11}(E_{31}E_{61} \vee E_{51}) \vee E_{21}(E_{41}E_{51} \vee E_{61}) \tag{1.11}$$

Suppose that the failures of branches 3 and 4 are mutually dependent and the failures of other branches are independent of each other. Hence $\Phi_i = \{E_{i1}, E_{i2}\}$ for $i = 1, 2, 5, 6$. Then, the following events are defined newly:

E_{71} both branches 3 and 4 function

E_{72} branch 3 functions and 4 fails

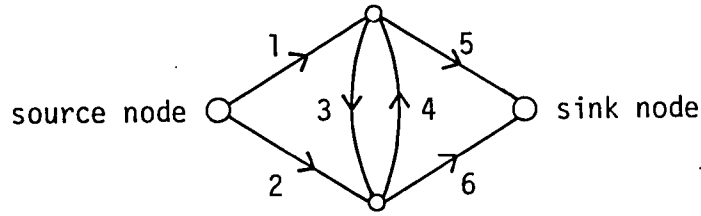


Fig. 3 Example 2 of System

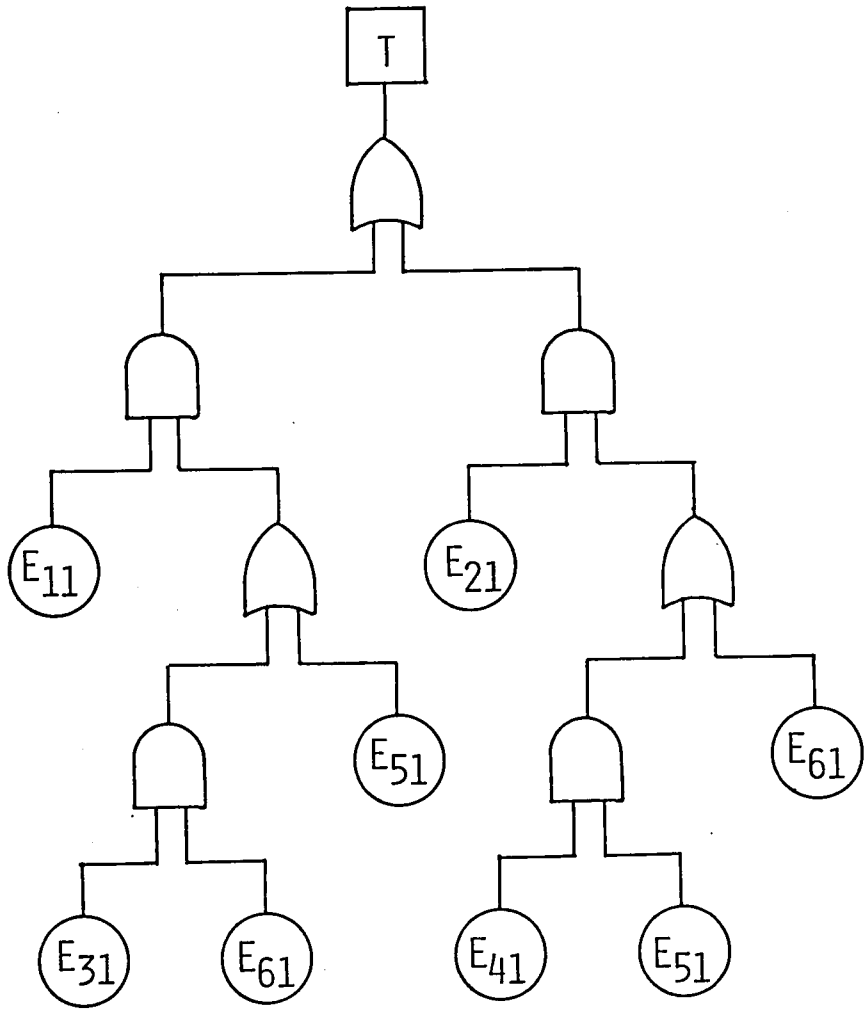


Fig. 4 Logic Tree for Example 2

E_{73} branch 3 fails and 4 functions

E_{74} both branches 3 and 4 fail

These events are mutually exclusive and $\Phi_7 = \{E_{71}, E_{72}, E_{73}, E_{74}\}$. E_{31}, E_{41} are equivalent to $E_{71} \vee E_{72}$, $E_{71} \vee E_{73}$, respectively. Therefore, Eq. (1.11) is transformed into the following equivalent Boolean structure function containing mutually exclusive primary events by replacing E_{31}, E_{41} with $E_{71} \vee E_{72}$, $E_{71} \vee E_{73}$, respectively.

$$\psi(\boldsymbol{E}) = E_{11}\{(E_{71} \vee E_{72})E_{61} \vee E_{51}\} \vee E_{21}\{(E_{71} \vee E_{73})E_{51} \vee E_{61}\} \tag{1.12}$$

Unpacking the right hand side of (1.12),

$$\chi(\boldsymbol{E}^*) = E_{11}E_{71}E_{61} \vee E_{11}E_{72}E_{61} \vee E_{11}E_{51} \vee E_{21}E_{71}E_{51} \vee E_{21}E_{73}E_{51} \vee E_{21}E_{61}. \tag{1.13}$$

$\psi(\boldsymbol{E})$ is coherent since $\chi(\boldsymbol{E}^*)$ contains only primary events belonging to $\Omega_i = \{E_{i1}\}$ for $i = 1, 2, 5, 6$ and $\Omega_7 = \{E_{71}, E_{72}, E_{73}\}$.

$\psi(\boldsymbol{E})$ has 6 minimal cut sets $C_{11} = \{E_{11}, E_{71}, E_{61}\}$, $C_{12} = \{E_{11}, E_{51}\}$, $C_{13} = \{E_{21}, E_{71}, E_{51}\}$, $C_{14} = \{E_{21}, E_{61}\}$, $C_{21} = \{E_{11}, E_{72}, E_{61}\}$, $C_{31} = \{E_{21}, E_{73}, E_{51}\}$, since each term of $\chi(\boldsymbol{E}^*)$ is not absorbed by any other term. They are partitioned into 3 sets $F_1 = \{C_{11}, C_{12}, C_{13}, C_{14}\}$, $F_2 = \{C_{21}\}$, $F_3 = \{C_{31}\}$.

Applying P4,

$$\Pr\{T = 1\} \leq 1 - (1 - p_{11}p_{71}p_{61})(1 - p_{11}p_{51})(1 - p_{21}p_{71}p_{51}) \times (1 - p_{21}p_{61}) + p_{11}p_{72}p_{61} + p_{21}p_{73}p_{51}.$$

1.4.2 Example of Noncoherent Structure

Example 3

Consider the electrical system, shown in Fig. 5, that has been presented by Fussell [21] and quoted by Bennetts [8]. The logic tree is shown in Fig. 6. The top event and primary events are:

T	no light (Z)
E_{11}	bulb fails (A)
E_{12}	bulb does not fail
E_{21}	power supply 1 fails (B)
E_{22}	power supply 1 operates
E_{31}	relay contact fails open (C)
E_{32}	relay contact does not fail open
E_{41}	circuit breaker fails open (D)
E_{42}	circuit breaker does not fail open
E_{51}	switch fails open (E)
E_{52}	switch fails closed (F)
E_{53}	switch normally operates
E_{61}	power supply 2 fails or relay coil open circuit or circuit breaker open circuit ($G \vee H \vee I$)
E_{62}	E_{61} does not occur

The notation in round brackets is used in Bennetts' paper [8]. Fig. 4 differs slightly from the corresponding fault trees in [8, 21]. The difference is due to an apparent misunderstanding of following facts. E_{51} and E_{52} are mutually exclusive. E_{52} and E_{62} must both occur in order that emf is not removed from circuit

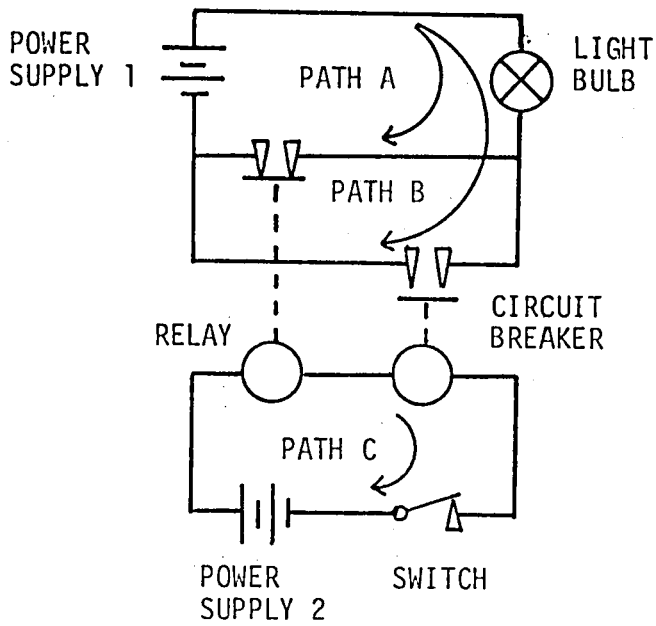


Fig. 5 Example 3 of System

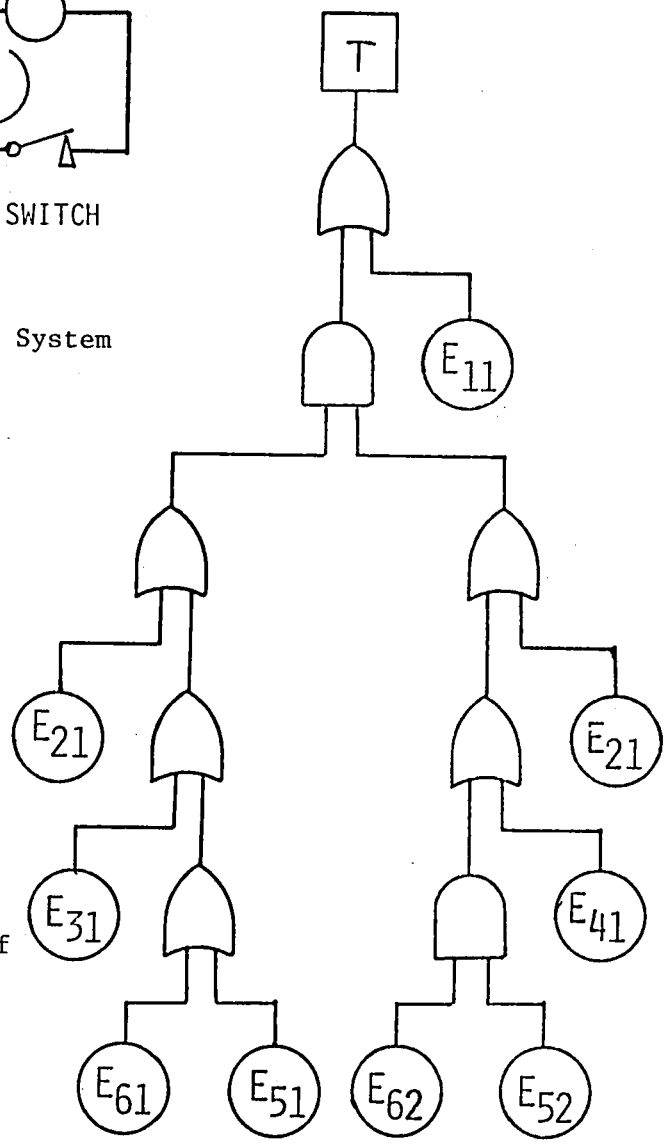


Fig. 6 Logic Tree of Example 3

path C. Emf is removed from circuit path C regardless of the state of switch if E_{61} occurs.

For the logic tree of Fig. 6,

$$\psi(\boldsymbol{E}) = E_{11} \vee (E_{21} \vee E_{31} \vee E_{51} \vee E_{61})(E_{21} \vee E_{41} \vee E_{52}E_{62}). \tag{1.14}$$

$\psi(\boldsymbol{E})$ is noncoherent since all elements of $\Phi_6 = \{E_{61}, E_{62}\}$ are relevant to $\psi(\boldsymbol{E})$. Φ_6 is complete set.

Apply Bayes theorem to a complete set Φ_i of noncoherent structure function $\psi(\boldsymbol{E})$.

$$\Pr\{T = 1\} = \Sigma_{j=1}^{\alpha_i} p_{ij} \Pr\{\psi(1_{ij}, \boldsymbol{E}) = 1\}. \tag{1.15}$$

$\psi(1_{ij}, \boldsymbol{E})$ does not contain any element of Φ_i. Thus the analysis of noncoherent structure function can be transformed into the analysis of coherent structure functions by applying Bayes theorem successively to all complete sets. This fact is illustrated below by the present example.

Apply Bayes theorem to Φ_6.

$$\Pr\{T = 1\} = p_{61} \Pr\{\psi(1_{61}, \boldsymbol{E}) = 1\} + p_{62} \Pr\{\psi(1_{62}, \boldsymbol{E}) = 1\} \tag{1.16}$$

$$\psi(1_{61}, \boldsymbol{E}) = E_{11} \vee E_{21} \vee E_{41}$$

$$\psi(1_{62}, \boldsymbol{E}) = E_{11} \vee (E_{21} \vee E_{31} \vee E_{51})(E_{21} \vee E_{41} \vee E_{52})$$

Both $\psi(1_{61}, \boldsymbol{E})$ and $\psi(1_{62}, \boldsymbol{E})$ are coherent.

Apply P4 to $\psi(1_{61}, \boldsymbol{E})$.

$$\Pr\{\psi(1_{61}, \boldsymbol{E}) = 1\} = 1 - (1 - p_{11})(1 - p_{21})(1 - p_{41}) \tag{1.17}$$

We obtain 5 minimal cut sets $C_{11} = \{E_{11}\}$, $C_{12} = \{E_{21}\}$, $C_{13} = \{E_{31}, E_{41}\}$, $C_{14} = \{E_{41}, E_{51}\}$, $C_{21} = \{E_{31}, E_{52}\}$ of $\psi(1_{62}, \boldsymbol{E})$ by applying P2. They are partitioned into two sets $F_1 = \{C_{11}, C_{12}, C_{13}, C_{14}\}$, $F_2 = \{C_{21}\}$. Apply P4.

$$\Pr\{\psi(1_{62}, \boldsymbol{E}) = 1\}$$

$$\leq 1 - (1 - p_{11})(1 - p_{21})(1 - p_{31}p_{41})(1 - p_{41}p_{51}) + p_{31}p_{52} \tag{1.18}$$

Apply (1.17), (1.18) to (1.16).

$$\Pr\{T = 1\} \leq p_{61}\{1 - (1 - p_{11})(1 - p_{21})(1 - p_{41})\}$$

$$+ p_{62}\{1 - (1 - p_{11})(1 - p_{21})(1 - p_{31}p_{41})(1 - p_{41}p_{51})$$

$$+ p_{31}p_{52}\}$$

This method requires applying Bayes theorem only to complete sets and reduces computation time. We can also decrease the number of resulting coherent structure functions by appropriately choosing the order of applying Bayes theorem to complete sets.

Chapter 2

Sequence Representation of Fault Tree

This chapter presents a representation of fault tree suitable for computer processing. A tree sequence is newly introduced as a represetation of fault tree; it represents completely the branching structure of tree. The properties and algorithms concerning the tree sequence are indicated. Various fault tree evaluations can be carried out through the combined use of tree sequence and reverse Polish sequence.

2.1 Introduction

According as the scale of fault tree becomes larger, the analysis becomes more complex and time-consuming. Therefore, a practical method of analysis by using a digital computer is required for the large scale fault tree.

When we analyse the fault tree by computer, the first problem we are faced with is "what is a representation of the fault tree useful for computer processing ? ". A list processing technique [8, 32] has been used in the analysis, which applies the reverse Polish sequence as a representation of fault tree. The use of the

reverse Polish sequence enables us to implement the arithmetic operations in a more elegant and programmable fashion [8], because it can be realised using an auxiliary stack, i.e., a last-in first-out pushdown list which is a conventional tool in the list processing programming technique. However, it is not sufficient to use only the reverse Polish sequence for processing the tree structure efficiently and executing the probabilistic evaluation of fault tree. In this chapter, a tree sequence is newly introduced to implement effectively the processing by using the recursive structure of tree. It is proved that the tree sequence represents completely the branching structure of tree, and some useful algorithms are obtained for processing the tree through the arithmetic operations of values composing the tree sequence. Various fault tree evaluations can be carried out through the combined use of tree sequence and reverse Polish sequence. Such techniques are illustrated in Chapters 3 and 4.

This chapter further introduces the extended definition of reverse Polish sequence for the fault tree containing k-out-of-n logic gates without transforming it into an equivalent fault tree containing only AND & OR gates. This is useful for implementing the probabilistic evaluation of fault tree more efficiently.

Section 2.2 defines basic terms concerning fault tree. Section 2.3 introduces reverse Polish sequence and tree sequence for the fault tree containing k-out-of-n gates. Section 2.4 discusses about some properties of tree sequence and Section 2.5 presents some algorithms concerning tree sequence. Section 2.6 explains how mutually exclusive primary events are treated in the sequence representation, when they are contained in the fault tree.

2.2 Fault Tree and Basic Definitions

The fault tree is a logic tree whose top event is a specific system failure. Each causative event that appears while proceeding from the top event to primary events is said an intermediate event. Each branch point is represented by a logic gate corresponding to its branching relation: AND gate, OR gate, k-out-of-n (k/n) gate, etc., are used there.

The system shown in Fig. 7 is a visual alarm circuit for overtemperature [27]. This system employs three temperature sensors which use bimetalic strips. If a sensor catches overtemperature, the attached switch is closed. Three switches SW_1, SW_2, SW_3 are connected with 2-out-of-3 structure, i.e., if at least two of three switches are closed, relay K is energized. Two contacts C_1, C_2 of relay K are connected with alarm lamps L_1, L_2, respectively. The system is good if either one alarm lamp lights when overtemperature occurs. It is assumed that the 2/3 detector is completely reliable.

A fault tree for this alarm circuit is shown in Fig. 8. The top event is "Overtemperature alarm system does not indicate overtemperature". This fault tree has 16 primary events and 8 intermediate events. Their list is given in Table 1. Primary events are represented by numbers 1, 2, ..., 16, top event by 1001 and intermediate events by 1002, 1003, ..., 1009.

Each causative event that is obtained directly by branching from each intermediate event or top event i through the logic gate Δ_i is said a *son* of event i, event i being the *father*, and the set of all sons of event i is denoted by SO(i). Δ_i is said the logic gate of event i. Event i is the output of Δ_i, and the sons of event i are the inputs of Δ_i.

A sequence of events $e_1 e_2 \ldots\ldots e_h$ is said a *path* from e_1 to e_h if $e_2 \in SO(e_1)$, $e_3 \in SO(e_2)$, ..., $e_h \in SO(e_{h-1})$. When a path from e_a to e_d exists, e_a is said an *ancestor* of e_d and e_d is said a *descendant* of e_a. Each event belongs to a *generation* as indicated in Fig. 8.

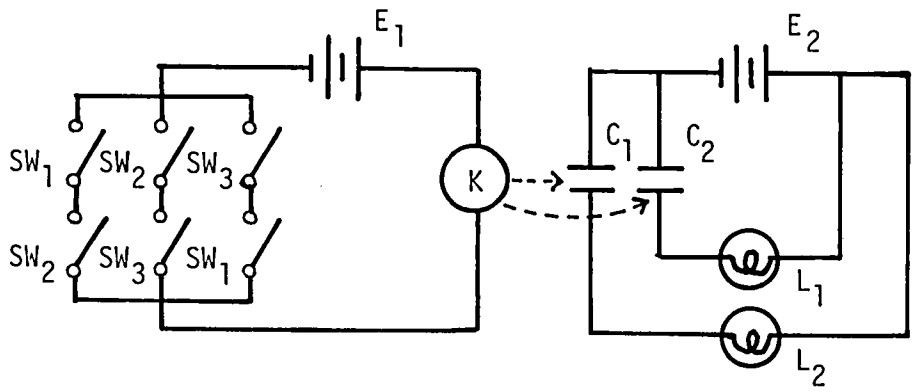


Fig. 7 Visual Alarm Circuit for Overtemperature

Table 1 List of Events

	Event Number	Event
Top Event	1001	Overtemperature alarm system does not indicate overtemperature
Intermediate Events	1002	Both alarm lamps L_1 and L_2 does not light
	1003	Coil of relay K is not energized
	1004, 1005	Alarm lamp L_1 (L_2) does not light
	1006	Temperature-sensing switches fail to close
	1007,1008,1009	Temperature-sensing switch SW_1 (SW_2, SW_3) fails to close
Primary Events	1, 2	Alarm lamp L_1 (L_2) fails
	3, 4	Contact C_1 (C_2) fails mechanically
	5, 6	Open wire in L_1 (L_2) circuit
	7, 10	Power supply E_1 (E_2) fails
	8	Coil of relay K fails (open)
	9	Open wire in relay circuit
	11, 12, 13	Switch SW_1 (SW_2, SW_3) fails mechanically (open)
	14, 15, 16	Bimetalic strip of temperature sensor connected with SW_1 (SW_2, SW_3) fails

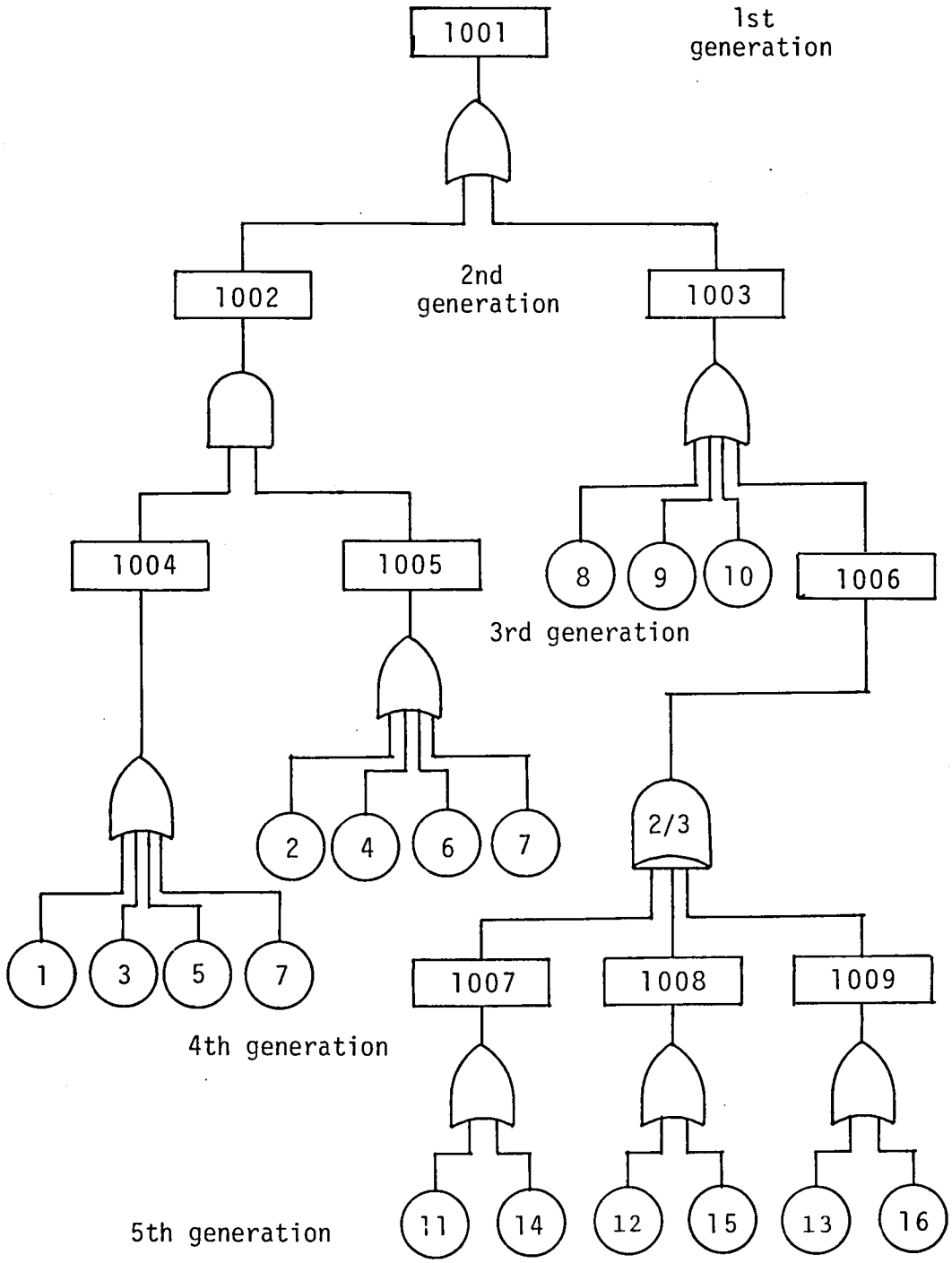


Fig. 8 Fault Tree for Visual Alarm System of Fig. 7

For the fault tree of Fig. 8, Event 1003 is the father of events 8, 9, 10, 1006. Events 8, 9, 10, 1006 are the sons of 1003, i.e., SO(1003) = {8, 9, 10,1006}, etc. The sequence 1001 1003 1006 1008 12 is a path. Event 1001 is an ancestor of 12, and 12 is a descendant of 1001, etc.

Let T, E_i be 0-1 binary variables having the value 1 or 0 corresponding to the occurrence or non-occurrence of the top event and primary event i, respectively. Then, T is represented as a Boolean function of primary events E_1, E_2, ... For the fault tree of Fig. 8,

$$T = (E_1 \vee E_3 \vee E_5 \vee E_7)(E_2 \vee E_4 \vee E_6 \vee E_7) \vee (E_8 \vee E_9 \vee E_{10} \vee 2/3(E_{11} \vee E_{14}, E_{12} \vee E_{15}, E_{13} \vee E_{16})),$$

where $2/3(a, b, c)$ denotes 2-out-of-3 logic, i.e., $2/3(a, b, c)$ is 1 if at least two of a, b, c are 1 and 0 otherwise.

A fault tree is said a *binary fault tree* if the number of sons of each intermediate event (or the top event) is only two.

A fault subtree of each intermediate event (or the top event) i is defined as that portion of the original fault tree which is developed below a son of i.

2.3 Reverse Polish Sequence and Tree Sequence

Let e_{ij} be sons of an intermediate event (or the top event) i; j = 1, 2, ..., l, rightward from the leftmost son in order. Then, let Q(i) be the sequence obtained by arranging l sons and logic gate Δ_i of i as follows:

$$Q(i) = e_{i1}\ e_{i2}\ e_{i3} \cdots\cdots\ e_{il}\ \Delta_i. \qquad (2.1)$$

In the computer processing, Δ_i is represented by an negative integer as shown in Table 2. The integers for k-out-of-n gate are also listed for small values of k and n in Table 3.

Table 2 Representation of Logic Gates

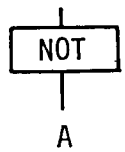
Logic Gate	Symbol in Fault Tree	Symbol in Boolean Function	Representation in Computer Processing
AND Gate		$\wedge$	-2
OR Gate		$\vee$	-3
k/n Gate		k/n	$-\left(\frac{n^2-5n+8}{2}+k+1\right)$
NOT Gate		$\bar{A}$	-1

Table 3 Values Assigned to Represent k/n Gates in Computer Processing

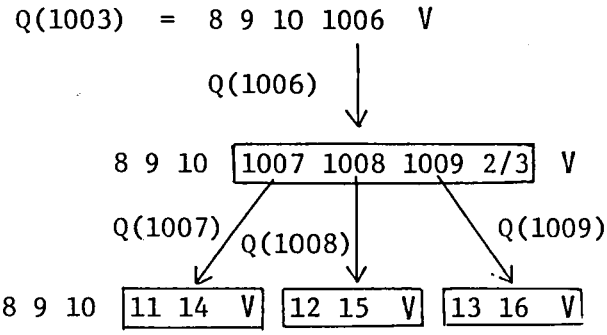
n	k	Assigned Value	n	k	Assigned Value
3	2	-4	6	2	-10
4	2	-5		3	-11
	3	-6		4	-12
5	2	-7		5	-13
	3	-8	7	2	-14
	4	-9		3	-15

For the fault tree of Fig. 8,

Q(1001) = 1002 1003 V
Q(1002) = 1004 1005 Λ
Q(1003) = 8 9 10 1006 V
Q(1004) = 1 3 5 7 V
Q(1005) = 2 4 6 7 V
Q(1006) = 1007 1008 1009 2/3
Q(1007) = 11 14 V
Q(1008) = 12 15 V
Q(1009) = 13 16 V

If event e_{ij} in Q(i) is an intermediate event, then substitute sequence Q(e_{ij}) for element e_{ij} of Q(i). If intermediate event e_t is contained further in the sequence obtained by the above substitutions, then substitute again the sequence Q(e_t) for event e_t. By repeating this process, we can obtain the sequence P(i) that contains only primary events and logic gates. This sequence P(i) is named as the *reverse Polish sequence* of i [8].

Applying the above technique to event 1003 of the fault tree of Fig. 8,



(2.2)

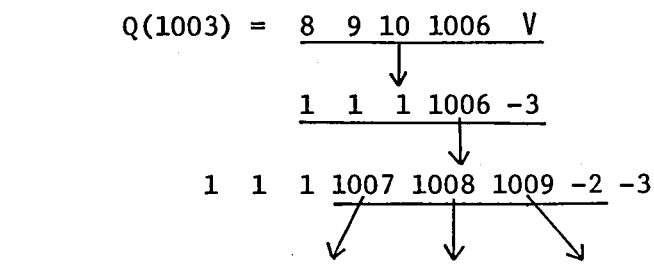
Similarly P(1002) and P(1001) are obtained:

P(1002) = 1 3 5 7 V 2 4 6 7 V Λ

P(1001) = 1 3 5 7 V 2 4 6 7 V Λ 8 9 10 11 14 V 12 15 V
13 16 V 2/3 V V

Now we can obtain the sequence TS(i) by replacing each primary event with value +1 (named as *leaf element*) and each logic gate having l sons with value $-(l-1)$ (named as *node element*) in the process of obtaining reverse Polish sequence P(i). This sequence TS(i) is named as the *tree sequence* of i.

Applying this technique to event 1003 of Fig. 8,



(2.3)

Similarly TS(1002) and TS(1001) are obtained:

TS(1002) = 1 1 1 1 -3 1 1 1 1 -3 -1

TS(1001) = 1 1 1 1 -3 1 1 1 1 -3 -1 1 1 1 1 1 -1 1 1 -1
1 1 -1 -2 -3 -1

The tree sequence of each event of binary fault tree contains only +1 and -1.

The following relations exist for P(i) and TS(i).

$$P(i) = P(e_{i1})\ P(e_{i2}) \ldots\ldots P(e_{il})\ \Delta_i \tag{2.4}$$

$$TS(i) = TS(e_{i1})\ TS(e_{i2}) \ldots\ldots TS(e_{il})\ -(l-1) \tag{2.5}$$

Additionally the *event sequence* ES(i) is defined as the sequence obtained by replacing each logic gate with its output in the process of obtaining reverse Polish sequence P(i).

For event 1003 of Fig. 8,

ES(1003) = 8 9 10 11 14 1007 12 15 1008 13 16 1009 1006 1003

2.4 Properties Concerning Tree Sequence

The tree sequence has some important properties such as mentioned below.

(P.1) When the values of elements of the tree sequence are added in order from the first element to the last element,
1) the sum never becomes 0 or negative while adding, and
2) the total sum, which is called the sum of tree sequence, is +1.

This property is checked by tree sequence TS(1003) for the fault tree of Fig. 8 (see Eq. (2.3)).

$$\begin{array}{cccccccccccccccc} 1 & + & 1 & + & 1 & + & 1 & + & 1 & + & (-1) & + & 1 & + & 1 & + \\ (+1) & & (+2) & & (+3) & & (+4) & & (+5) & & (+4) & & (+5) & & (+6) & \end{array}$$

$$\begin{array}{cccccccccc} (-1) & + & 1 & + & 1 & + & (-1) & + & 1 & + & 1 \\ (+5) & & (+6) & & (+5) & & (+6) & & (+7) & & \end{array}$$

$$\begin{array}{cccccccc} + & (-1) & + & (-2) & + & (-3) & = & +1 \\ & (+6) & & (+4) & & & & \end{array}$$

The value in the parenthesis under each element is the sum from the first element to that element; certainly any value is not 0 or negative.

(P.2) In a tree sequence with two or more elements, the sequence obtained by removing the last element $-m$ (the last element is always a node element) can be partitioned into $m+1$ tree subsequences and the way of partition is unique. Accordingly the tree sequence represents uniquely a tree that has each leaf element as a leaf and each node element as a node.

Each tree subsequence obtained by the partition of P.2 is said a *son of the tree sequence*, and the tree sequence is then the father of each tree subsequence.

TS(1003) of Eq. (2.3) has four sons S_1, S_2, S_3, S_4 as follows:

$$\underbrace{1}_{S_1} \quad \underbrace{1}_{S_2} \quad \underbrace{1}_{S_3} \quad \underbrace{1\ 1\ -1\ 1\ 1\ -1\ 1\ 1\ -1\ -2}_{S_4} \quad -3$$

It can easily be checked that each of S_1, S_2, S_3, S_4 also has P.1. The algorithm for obtaining the sons of tree sequence is given later (see A.2 of Section 2.5).

The proofs of P.1 and P.2 are given in Appendix B. These properties imply that a tree sequence represents completely the branching structure of tree and reveals many kinds of properties of tree. The analysis of fault tree is therefore reduced to the processing of reverse Polish sequence and tree sequence.

From P.2, the following facts are verified:

(i) When the sequence obtained by removing the last element $-(l-1)$ from TS(i) in Eq.(2.5) is partitioned into l sons of TS(i) according to P.2, those sons are TS(e_{i1}), TS(e_{i2}), ..., TS(e_{il}) in order from the leftmost son, and the corresponding sequences of P(i) are P(e_{i1}), P(e_{i2}), ..., P(e_{il}), respectively.

(ii) Suppose we have a sequence composed of leaf elements and node elements and it has properties 1) and 2) described in P.1. Then we can uniquely compose a tree corresponding to that sequence.

The following properties of tree sequence are also obtained from the properties of a tree.

(P.3) 1) The tree sequence with length one is 1. 2) The tree sequence with length two is 1 0. 3) The tree sequence with the length more than two necessarily begins with 1 1 or 1 0 , and ends with a node element.

(P.4) There exists only one tree subsequence that has a node element of tree sequence as the last element. The last element of a tree subsequence is named as the *representative element* of that tree subsequence.

(P.5) The number of tree subsequences of the tree sequence is equal to the length of the tree sequence.

(P.6) The sequence obtained by replacing a tree subsequence with another tree sequence is also a tree sequence.

(P.7) The sequence obtained by removing a tree subsequence

from the tree sequence and then replacing the representative element $-m$ of the father of that tree subsequence with $-(m-1)$ (removing it if $m = 0$) is also a tree sequence.

(P.8) The first tree subsequence of the tree sequence is defined as the tree subsequence 1 1 1...1 $-m$ (the number of leaf elements is $m+1$) whose representative element is the first node element $-m$ of the tree sequence. The tree sequence is reduced to only one leaf element (tree sequence with length one) by recursively repeating the operation of replacing the first tree subsequence with one leaf element 1.

2.5 Algorithms Concerning Tree Sequence

Some effective algorithms for processing the tree sequence are obtained by using the properties mentioned in Section 2.4.

(A.1) Algorithm for obtaining the tree subsequence having as the representative element a node element $-\nu$ of the tree sequence.

Add the value of each element successively to the left starting at $-\nu$ (including $-\nu$). If the sum becomes +1 for the first time at a leaf element while adding, then the subsequence from that leaf element to $-\nu$ is the tree subsequence to be required.

This algorithm is illustrated by TS(1003) of Eq. (2.3).

$$TS(1003) = 1\ 1\ 1\ 1\ 1\ -1\ 1\ 1\ -1\ 1\ 1\ -1\ \underline{\textcircled{-2}\ -3}$$

Suppose that we want to obtain the tree subsequence having the circled element -2 as the representative element. Then, add the value of each element successively to the left starting at element -2 by following A.1.

$$\begin{array}{ccccccccccccc} (-2) & + & (-1) & + & 1 & + & 1 & + & (-1) & + & 1 & + & 1 & + & (-1) & + & 1 & + & 1 & = +1 \\ & & (-3) & & (-2) & & (-1) & & (-2) & & (-1) & & (0) & & (-1) & & (0) & & \end{array}$$

The value in the parenthesis under each element is the sum from the first to that element. The sum becomes +1 for the first time at the fourth element of TS(1003). Thus the underlined part is the tree subsequence to be required.

(A.2) Algorithm for obtaining sons, S_1, S_2, ..., S_l, of TS(i) (The last element of TS(i) is -(l-1).):

Step 1: Set $q \leftarrow l$, then let R_l be the sequence obtained by removing the last element of TS(i), and go to Step 2.

Step 2: By using A.1, obtain the tree subsequence whose representative element is the last element of R_q, then put this tree subsequence as S_q, and go to Step 3.

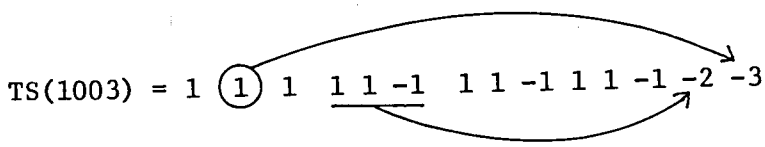
Step 3: Let R_{q-1} be the sequence obtained by removing S_q from R_q. If $q = 2$, let $S_1 = R_{q-1}$ and stop. If $q \neq 2$, set $q \leftarrow q - 1$ and go to Step 2.

It can easily be verified that four sons of TS(1003) indicated in Section 2.4 are obtained by using A.2.

(A.3) Algorithm for obtaining the representative element of father of a tree subsequence:

Add the value of each element successively to the right starting at the next element of the representative element of the tree subsequence. If the sum becomes 0 or negative for the first time at a node element while adding, then that node element is the representative element to be required.

For TS(1003) of Eq. (2.3), the representative elements of fathers of the circled element and underlined tree subsequence are shown below by the arrows, respectively.



Each representative element is obtained by applying A.3 as follows. For the circled element,

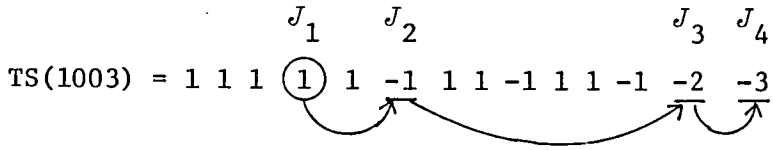
$$1 + 1 + 1 + (-1) + 1 + 1 + (-1) + 1 + 1 + (-1) + (-2) + (-3)$$
$$(2)\ \ (3)\ \ (2)\ \ (3)\ \ (4)\ \ (3)\ \ (4)\ \ (5)\ \ (4)\ \ (2)$$
$$= -1$$

The value in the parenthesis under each element is the sum from the first to that element. For the underlined tree subsequence,

$$1 + 1 + (-1) + 1 + 1 + (-1) + (-2) = 0$$
$$(2)\ \ (1)\ \ (2)\ \ (3)\ \ (2)$$

Let C_1 be a tree subsequence of the tree sequence and J_1 be the representative element of C_1, C_2 be the father of C_1 and J_2 be the representative element of C_2, and then C_3 be the father of C_2 and J_3 be the representative element of C_3. When the above process is repeated successively, the sequence $J_1\ J_2\ J_3$ ..... is named as the *ascending path* from J_1. The ascending path from J_1 is obtained by applying A.3 iteratively.

The ascending path from the circled element of TS(1003) of Eq. (2.3) are shown below by the arrows.



There are cases of redesigning a system, such as adding some subsystems newly, replacing a subsystem with another subsystem and removing some subsystems. In such cases, we can easily obtain P(1001), TS(1001) and ES(1001) for top event 1001 of fault tree of the redesigned system from those of the original system as shown in the following methods (1), (2), (3).

(1) Adding a son s to the sons of event i.

Insert ES(s) just before i in event sequence ES(1001) of top event 1001. Insert P(s) and TS(s) just before the element corresponding to event i in P(1001) and TS(1001), respectively. Then replace with $-(\nu+1)$ the node element $-\nu$ of TS(1001) corresponding to event i.

(2) Replacing an intermediate event i with another intermediate event i'.

Replace ES(i), P(i) and TS(i) with ES(i'), P(i') and TS(i'), respectively.

(3) Removing an event i from the fault tree.

Remove ES(i), P(i) and TS(i) from ES(1001), P(1001) and TS(1001), respectively. Then, replace the representative element $-m$ of the father of TS(i) with $-(m-1)$ if $m \geq 2$. If $m = 1$, remove $-m$ and the corresponding elements of ES(1001) and P(1001).

We can transform P(i) and TS(i) of event i into the equivalent shorter sequences by the following method.

(4) Shortening reverse Polish sequence and tree sequence.

Let S be a tree subsequence of TS(i) and $-m$ be the representative element of S. Suppose that the element of P(i) corresponding to $-m$ is AND-gate (or OR-gate).

Find AND-gates (or OR-gates) G_1, G_2, ..., G_q among the representative elements of the sons of S, and suppose that the elements of TS(i) corresponding to those gates are $-m_1$, $-m_2$, ..., $-m_q$, respectively. Then, we can obtain the equivalent sequences by removing G_1, G_2, ..., G_q from P(i) and $-m_1$, $-m_2$, ..., $-m_q$ from TS(i), and then changing $-m$ into $-(m + m_1 + m_2 + ... + m_q)$.

This method is illustrated for the examples of P(i) and TS(i) such as shown below.

G_1 G_2

P(i) = 1 2 ∨ 3 4 5 ∨ 7 8 ∧ 9 6 10 ∨ ∨ ∧

TS(i) = 1 1 -1 (1) (1 1 -1) (1 1 -1) (1 1 1 -2) -3 -1

S

The element of P(i) corresponding to representative element -3 of tree subsequence S of TS(i) is OR-gate (represented by -3). There exist two OR-gates G_1, G_2 among the representative elemenets of four sons (circled) of S. Because m = 3, m_1 = 1 and m_2 = 2, $m + m_1 + m_2 = 6$. Accordingly we can obtain the following equivalent sequences.

P(i) = 1 2 ∨ 3 4 5 7 8 ∧ 9 6 10 ∨ ∧

TS(i) = 1 1 -1 1 1 1 1 1 -1 1 1 1 -6 -1

(5) Transforming the sequences for the fault tree with only AND-gates and OR-gates into those for an equivalent binary tree.

Replace each node element $-\nu$ ($\nu \geq 2$) with sequence -1 -1 ....-1 (length ν) in tree sequence TS(1001) of top event 1001, and then replace the logic gate of P(1001) corresponding to $-\nu$ with the sequence composed of ν logic gates of the same kind. The sequences obtained are the tree sequence and reverse Polish sequence of the top event of an equivalent binary fault tree.

If the fault tree has P(i) and TS(i) given at the beginning of this page as P(1001) and TS(1001), then the sequences for an equivalent binary fault tree are

P(1001) = 1 2 ∨ 3 4 5 ∨ 7 8 ∧ 9 6 10 ∨ ∨ ∨ ∨ ∨ ∧

TS(1001) = 1 1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 -1 -1 -1

2.6 Treatment of Mutually Exclusive Primary Events

This section explains how mutually exclusive primary events are treated in the sequence representation, when they appear in the fault tree.

Consider an exhaustive set Φ_i of α_i mutually exclusive primary events. They have the same event number i and each of them is assigned another index number j, $j = 1, 2, \ldots, \alpha_i$, i.e., they are represented by $(i, 1)$, $(i, 2)$, $\ldots$, (i, α_i) (note that they are represented by $E_{i1}, E_{i2}, \ldots, E_{i\alpha_i}$, respectively, in Chapter 1). Let γ_i be the number of primary events in Φ_i which appear in the fault tree; $\gamma_i \leq \alpha_i$. If $\gamma_i = 1$, then primary event (i, j) appearing in the fault tree can be represented by only event number i.

Then, in reverse Polish sequence $P(T)$ of top event T, all events in Φ_i are represented by event number i. Another sequence $MS(T)$ is prepared to distinguish each of them from others; $MS(T)$ has the index number j as the element corresponding to event (i, j) if $\gamma_i \geq 2$ and has the value 0 as the element corresponding to each logic gate or primary event (i, j) in the case of $\gamma_i = 1$.

For the fault tree of Fig. 1 in Chapter 1, the sequences are obtained as follows.

$$\begin{aligned} P(T) &= 1\ 2\ 3\ \vee\ \wedge\ 1\ 2\ 3\ \wedge\ \vee\ \vee \\ TS(T) &= 1\ 1\ 1\ -1\ -1\ 1\ 1\ 1\ -1\ -1\ -1 \\ MS(T) &= 2\ 2\ 2\ 0\ 0\ 3\ 3\ 3\ 0\ 0\ 0 \end{aligned} \tag{2.6}$$

For the fault tree of Fig. 5 in Chapter 1, the sequences are obtained as follows, since $\gamma_1 = \gamma_2 = \gamma_3 = \gamma_4 = 1$ and $\gamma_5 = \gamma_6 = 2$.

$$\begin{aligned} P(T) &= 2\ 3\ 6\ 5\ \vee\ \vee\ \vee\ 6\ 5\ \wedge\ 4\ \vee\ 2\ \vee\ \wedge\ 1\ \vee \\ TS(T) &= 1\ 1\ 1\ 1\ -1\ -1\ -1\ 1\ 1\ -1\ 1\ -1\ 1\ -1\ -1\ 1\ -1 \\ MS(T) &= 0\ 0\ 1\ 1\ 0\ 0\ 0\ 2\ 2\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \end{aligned} \tag{2.7}$$

To reduce storage requirements, sequences TS(T) and MS(T) can be maken into one sequence TMS(T) which has as the qth element the sum of the qth elements of TS(T) and MS(T).

For the sequences of (2.6),

$$\text{TMS}(T) = 3\ 3\ 3\ -1\ -1\ 4\ 4\ 4\ -1\ -1\ -1.$$

For the sequences of (2.7),

$$\text{TMS}(T) = 1\ 1\ 2\ 2\ -1\ -1\ -1\ 3\ 3\ -1\ 1\ -1\ 1\ -1\ -1\ 1\ -1.$$

Chapter 3

An Efficient Bottom-up Algorithm for Enumerating Minimal Cut Sets of Fault Tree

This chapter improves the conventional bottom-up algorithm for enumerating minimal cut sets of fault tree. The improvement is to reduce the number of checks of redundant terms for the logical product of two reduced sum-of-product forms. The algorithm for executing this process is presented and illustrated by an example. The computational results for several examples are presented to demonstrate the efficiency of the algorithm.

3.1 Introduction

As stated in Chapter 1, the top event or an intermediate event can be represented as a Boolean function of primary events derived from that event, by treating the symbol attached to each event as a binary variable having the value 1 or 0 corresponding to the state that the event has or has not occurred. If the Boolean function for the top event is coherent, the reduced sum-of-product (s. o.p.) form is minimal and the set of primary events appearing in each term is a minimal cut set. It means that all primary events of a minimal cut set must occur at a minimum for the top event to

occur. The list of all minimal cut sets is useful for various probabilistic evaluations, the determination of critical path to the occurrence of system failure, etc. [5 , 20, 70] If the Boolean function for the top event is not coherent, the reduced s.o.p. form may not be minimal, but it is still a symplified s.o.p. form useful for fault tree evaluation.

Some algorithms have been presented for obtaining a reduced s.o.p. form for the top event. They can be classified into two groups: bottom-up [8, 72] and top-down [17, 28, 61]. A bottom-up algorithm begins with primary events and works upward to the top event while a top-down algorithm begins with the top event and works downward to primary events. Either algorithm is based on the principle of discarding redundant terms from a s.o.p. form to yield the reduced s.o.p. form.

Generally speaking, it takes too much computation time for obtaining all minimal cut sets, as the scale of fault tree becomes large. This chapter aims to improve the conventional bottom-up algorithm [8] so as to obtain all minimal cut sets more quickly. The improvement is to reduce the number of checks of redundant terms for the logical product of two reduced s.o.p. forms. It is proved that, when the logical product of two reduced s.o.p. forms is expanded by the distribution rule, one need only check if each resulting term is absorbed by some terms of two original s.o.p. forms. Only the algorithm for executing this process is presented in this chapter; the entire computer program to obtain all minimal cut sets is given in [55].

Section 3.2 is a general view of algorithm and the detailed algorithm is given in Section 3.3. Section 3.4 illustrates the algorithm by an example and Section 3.5 presents the computational results for several examples to demonstrate the efficiency of this algorithm.

3.2 Preliminaries and General View

3.2.1 Preliminaries

Some terms and notation used in this chapter are defined below. Basic terms on fault tree defined in the former chapters are used in succession.

reduced s.o.p. form — s.o.p. form obtained by discarding redundant terms from a s.o.p. form of Boolean function by applying the idempotence rule ($X \vee X = X$) and absorption rule ($X \vee XY = X$).

T_k — reduced s.o.p. forms for two events; $k = 1, 2$.

n_k — number of terms of T_k.

i, j — dummy index for terms of T_k.

c_{ki} — set of primary events appearing in term i of T_k.

C_k — $\{c_{k1}, c_{k2}, \ldots, c_{kn_k}\}$.

$A \otimes B$ — whole set of unions of an element of A and an element of B for two sets A, B each element of which is a set of primary events, e.g., $C_1 \otimes C_2 = \{c_{11} \cup c_{21}, \ldots, c_{11} \cup c_{2n_2}, c_{12} \cup c_{21}, \ldots, c_{12} \cup c_{2n_2}, \ldots, c_{1n_1} \cup c_{21}, \ldots, c_{1n_1} \cup c_{2n_2}\}$.

c — set of primary events currently being checked; $c \in C_1 \otimes C_2$.

i^*, j^* — index number for term of T_1 or T_2 that yields c, i.e., $c = c_{1i^*} \cup c_{2j^*}$.

T — reduced s.o.p. form of logical product $T_1 T_2$.

C — set of sets of primary events, each being the set of primary events contained in a term of T.

ϕ — empty set.

common primary event primary event appearing in both T_1 and T_2.

non-common primary event primary event appearing in only one of T_1 and T_2.

absorb c_p absorbs c_q if $c_p \subset c_q$ for two sets c_p, c_q of primary events.

Additional notation is defined in Section 3.3.

In this chapter, the following assumptions are made:

1. Mutually exclusive primary events are allowed to appear in the fault tree.

2. The Boolean function for the top event need not to be coherent.

3. Only OR & AND gates are allowed. If logic gates, such as NOT, XOR, NAND, k-out-of-n, appear in the fault tree, then the algorithm can be applied after transforming it into an equivalent fault tree containing only OR & AND gates.

3.2.2 General View of Algorithm

The algorithm begins with primary events and repeat, until reaching the top event, the process of expanding the logical product (for AND gate) or sum (for OR gate) of reduced s.o.p. forms for two causative events into a s.o.p. form by the distribution rule and then discarding redundant terms by applying the idempotence and absorption rules to yield an equivalent reduced s.o.p. form. This structure is known as a bottom-up algorithm [8, 72]. Our algorithm has also this structure.

The logical combination of two reduced s.o.p. forms having n_1 and n_2 terms yields a s.o.p. form having $n_1 \times n_2$ terms (for a product) or $n_1 + n_2$ terms (for a sum) by applying the distribution rule; thus there are a sharp increase of terms in the expansion of any combination. All pairs of terms must be checked against each other by applying the idempotence and absorption rules. Since, for

a product, the total number of pairs of n_1 X n_2 terms is n_1n_2 X $(n_1n_2 - 1)$, the number of checks by two rules becomes steeply greater as n_k increases. Thus the checking for products would dominate the computation time, especially for a fault tree having AND gates near the top. The conventional bottom-up algorithms [8, 72] do not take notice of this fact. Accordingly it is desired to shorten the computation time of expanding and checking for logical product; hence, our effort is focused on this point.

Consider the process of obtaining T from the logical product T_1T_2. Each primary event appearing in T_1, T_2 is classified into either common primary event or non-common primary event. The algorithm for obtaining C is based on the following principles.

1. If c contains only non-common primary events, then c is always an element of C. Thus it is not necessary to check c at all.

2. If c contains at least one common primary event, then only elements of a subset of $C_1 \cup C_2$ are required to check c.

The use of these principles appreciably decreases the number of checks. The theoretical limit to the number of checks by our algorithm is n_1n_2 X $(n_1 + n_2 - 1)$ while that by Bennetts' algorithm [8] is n_1n_2 X $(n_1n_2 - 1)$. The detailed algorithm is given in the next section.

3.3 Algorithm

3.3.1 Partial Algorithms

In this section, the algorithm, called ANCHEK, is presented for obtaining the reduced s.o.p. form T of logical product T_1T_2, i.e., yielding C from C_1, C_2. Then the algorithm (called ORCHEK) presented by Bennetts [8] for obtaining the reduced s.o.p. form of logical sum $T_1 \cup T_2$ is modified. These algorithms are the parts of the entire algorithm for obtaining all minimal cut sets.

ANCHEK Algorithm:

Step 1. Classify primary events appearing in C_1, C_2 into common and non-common primary events.

To do this, obtain set $(c_{11} \cup c_{12} \cup \ldots \cup c_{1n_1}) \cap (c_{21} \cup c_{22} \cup \ldots \cup c_{2n_2})$. The elements of this set are common primary events. Other primary events are non-common.

Step 2. Partition C_k (for $k = 1, 2$) into three sets C_{ka}, C_{kb}, C_{kc}.

C_{ka}: the set of elements of C_k containing only common primary events.

C_{kb}: the set of elements of C_k containing both common and non-common primary events.

C_{kc}: the set of elements of C_k containing only non-common primary events.

Step 3. Find the following sets from C_{1a}, C_{2a}, C_{1b}, C_{2b}.

F : the set of elements which are in both C_{1a} and C_{2a}.

F_{ka} (or F_{kb}): the set of elements of C_{ka} (or C_{kb}), each of which could be absorbed by at least one element of $C_{(3-k)a}$. Do for $k = 1, 2$.

Step 4. Let $C'_{ka} \equiv C_{ka} - (F \cup F_{ka})$, $C'_{kb} \equiv C_{kb} - F_{kb}$ for $k = 1, 2$. Then set C is obtained by the following 6 checking rules.

4.1 For $c \in (F \cup F_{1a} \cup F_{1b} \cup F_{2a} \cup F_{2b})$, $c \in C$.

4.2 For $c \in C'_{1a} \otimes C'_{2a}$, let $D_k \equiv C_{ka}$ for $k = 1, 2$ and check c by Subroutine RULEA.

4.3 For $c \in C'_{1a} \otimes C'_{2b}$ (or $c \in C'_{1b} \otimes C'_{2a}$), let $D_1 \equiv C_{1a}$ (or $D_1 \equiv C_{1a} \cup C_{1b}$), $D_2 \equiv C_{2a} \cup C_{2b}$ (or $D_2 \equiv C_{2a}$) and check c by Subroutine RULEA.

4.4 For $c \in C'_{1b} \otimes C'_{2b}$, let $D_k \equiv C_{ka} \cup C_{kb}$ for $k = 1, 2$ and check c by Subroutine RULEA.

4.5 For $c \in C_{1c} \otimes (C'_{2a} \cup C'_{2b})$ (or $c \in (C'_{1a} \cup C'_{1b}) \otimes C_{2c}$), let $D \equiv C_{1b}$ (or $D \equiv C_{2b}$) and check c by Subroutine RULEB.

4.6 For $c \in C_{1c} \otimes C_{2c}$, $c \notin C$ if c contains two or more mutually exclusive primary events. Otherwise $c \in C$.

Subroutine RULEA:

Step 1. If c contains two or more mutually exclusive primary events, then $c \notin C$ and return to ANCHEK. Otherwise set $k \leftarrow 1$, $i \leftarrow 1$ and go to Step 2.

Step 2. If $c_{ki} \in D_k$ and $c_{ki} \neq c_{1i^*}$ for $k = 1$ ($c_{ki} \neq c_{2j^*}$ for $k = 2$), then check c by c_{ki} using the following 4 sub-rules. Otherwise go to Step 3.

a. If $c_{ki} \subset c$ and $c_{ki} \cup c_{2j^*} \subset c$ for $k = 1$ ($c_{1i^*} \cup c_{ki} \subset c$ for $k = 2$), then $c \notin C$ and return to ANCHEK.

b. If $c_{ki} \subset c$ and $c_{ki} \cup c_{2j^*} = c$, $i < i^*$ for $k = 1$ ($c_{1i^*} \cup c_{ki} = c$, $i < j^*$ for $k = 2$), then $c \notin C$ and return to ANCHEK.

c. If $c_{ki} \subset c$ and $c_{ki} \cup c_{2j^*} = c$, $i > i^*$ for $k = 1$ ($c_{1i^*} \cup c_{ki} = c$, $i > j^*$ for $k = 2$), then store c_{ki} as an element of set S_{ki^*} for $k = 1$ (S_{kj^*} for $k = 2$) and go to Step 3.

d. If $c_{1i} \not\subset c$, then go to Step 3.

Step 3. If $k = 1$ and $i = n_1$, then set $k \leftarrow 2$, $i \leftarrow 1$ and go to Step 2. If $k = 2$ and $i = n_2$, then go to Step 4. Otherwise set $i \leftarrow i + 1$ and go to Step 2.

Step 4. If $S_{1i^*} \neq \phi$, $S_{2j^*} \neq \phi$ and c is not equal to at least one element of $S_{1i^*} \otimes S_{2j^*}$, then $c \notin C$. Otherwise $c \in C$. Return to ANCHEK.

Subroutine RULEB:

Step 1. If c contains two or more mutually exclusive primary events, then $c \notin C$ and return to ANCHEK. Otherwise go to Step 2.

Step 2. If c is absorbed by at least one element of D, then $c \notin C$. Otherwise $c \in C$. Return to ANCHEK.

The proofs of 6 checking rules in Step 4 of ANCHEK are given in Appendix C. The logical product T_1T_2 can and might contain mutually exclusive primary events. Mutually exclusive primary events need not be distinguished from non-mutually exclusive primary events, except that c is deleted if c contains two or more mutually exclusive primary events. (See Step 1 of Subroutines RULEA, RULEB.)

Upon transforming the logical sum of two reduced s.o.p. forms T_1, T_2 into an equivalent reduced s.o.p. form, ORCHEK algorithm presented by Bennetts [8] is modified so as to check each term of T_1 only by the terms of T_2: this treatment is proper because the terms in a reduced s.o.p. form do not absorb each other. This technique seems to be more effective than preordering the terms of $T_1 \vee T_2$ into an ascending order of number of primary events in each term.

Modified ORCHEK Algorithm:

Step 1. Set $i \leftarrow 1$, $j \leftarrow 1$.

Step 2. If $c_{1i} \subset c_{2j}$, then put c_{2j} in set S_2 and go to Step 3. If $c_{1i} \supset c_{2j}$, then put c_{1i} in set S_1 and go to Step 5. If $c_{1i} = c_{2j}$, then put c_{1i} in sets S_1, S_2, S and go to Step 5. Otherwise go to Step 5.

Step 3. If $j = n_2$, then go to Step 5. If $j \neq n_2$, then set $j \leftarrow j + 1$ and go to Step 4.

Step 4. If $c_{2j} \in S_2$, then go to Step 3. If $c_{2j} \notin S_2$, then go to Step 2.

Step 5. If $i = n_1$, then go to Step 6. If $i \neq n_1$, then set $i \leftarrow i + 1$.

If $c_{1i} \in S_1$, then repeat Step 5. If $c_{1i} \notin S_1$, then set $j \leftarrow 1$ and go to Step 4.

Step 6. Stop. Primary events contained in each element of set $(C_1 - S_1) \cup (C_2 - S_2) \cup S$ compose a term of the reduced s.o.p. form of $T_1 \vee T_2$.

3.3.2 Overall Algorithm

The basic operation of algorithm is to obtain the reduced s.o.p. form of the output of a gate from the reduced s.o.p. forms of its inputs. ANCHEK algorithm is applied for AND gate and modified ORCHEK algorithm for OR gate; since each algorithm operates on two reduced s.o.p. forms at a time, it is applied $(l - 1)$ times for a gate with l inputs.

The order of applying the above basic operation begins with the lowest gate of the fault tree and proceeds toward upper gates successively. This process is realized by the list processing technique using reverse Polish sequence and tree sequence presented in Chapter 2. This structure of overall algorithm is same as Bennetts' algorithm [8] in principle, except that Bennetts uses MULTIPLY & ORCHEK algorithms for AND gate and ORCHEK algorithm for OR gate as the basic operation. The flow chart of overall algorithm is shown in Fig. 9.

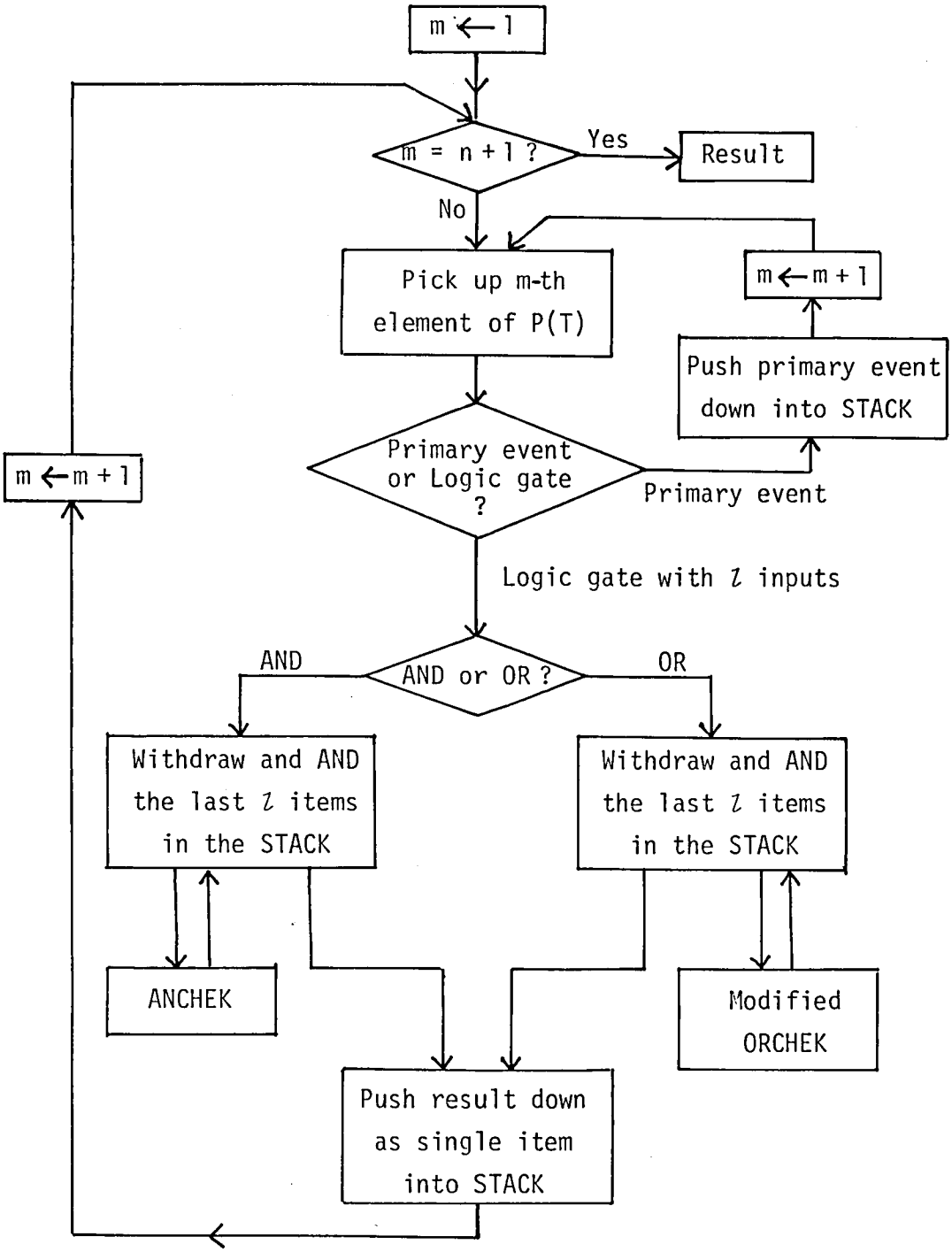


Fig. 9 Flow Chart of Overall Algorithm

3.4 Illustration of Algorithm by Example

ANCHEK algorithm presented in Section 3.3 is illustrated by the example of fault tree of Fig. 10 with 16 primary events which are represented by integers from 1 to 16.

Now consider the stage that the reduced s.o.p. forms T_1, T_2 for two intermediate events E_1, E_2 have been obtained.

The elements of C_1 are:

$c_{11} = \{2,5\}$, $c_{12} = \{3,6,10\}$, $c_{13} = \{10,16\}$, $c_{14} = \{2,3\}$,
$c_{15} = \{3,6,14\}$, $c_{16} = \{5,10,11\}$, $c_{17} = \{8,9,13\}$, $c_{18} = \{1,6\}$.

The elements of C_2 are:

$c_{21} = \{3, 6\}$, $c_{22} = \{4,12\}$, $c_{23} = \{1,6,14,15\}$, $c_{24} = \{7,8,13\}$,
$c_{25} = \{7,12,15\}$, $c_{26} = \{2,3\}$.

The results obtained by the steps of ANCHEK algorithm are shown below; in Step 4, the number in parens. is the number of checks by using RULEA or RULEB.

Step 1. The common primary events are 1, 2, 3, 6, 8, 13, 14 and the non-common primary events are 4, 5, 7, 9, 10, 11, 12, 15, 16.

Step 2. $C_{1a} = \{c_{14}, c_{15}, c_{18}\}$, $C_{1b} = \{c_{11}, c_{12}, c_{17}\}$, $C_{1c} = \{c_{13}, c_{16}\}$, $C_{2a} = \{c_{21}, c_{26}\}$, $C_{2b} = \{c_{23}, c_{24}\}$, $C_{2c} = \{c_{22}, c_{25}\}$.

Step 3. $F = \{c_{14}$ (equal $c_{26})\}$, $F_{1a} = \{c_{15}\}$, $F_{1b} = \{c_{12}\}$, $F_{2a} = \phi$, $F_{2b} = \{c_{23}\}$.

Step 4. $C'_{1a} = \{c_{18}\}$, $C'_{2a} = \{c_{21}\}$, $C'_{1b} = \{c_{11}, c_{17}\}$, $C'_{2b} = \{c_{24}\}$.

4.1 Let $c_{12} \in C$, $c_{14} \in C$, $c_{15} \in C$, $c_{23} \in C$.

4.2 $c_{18} \cup c_{21} = \{1,3,6\} \in C$, (3).

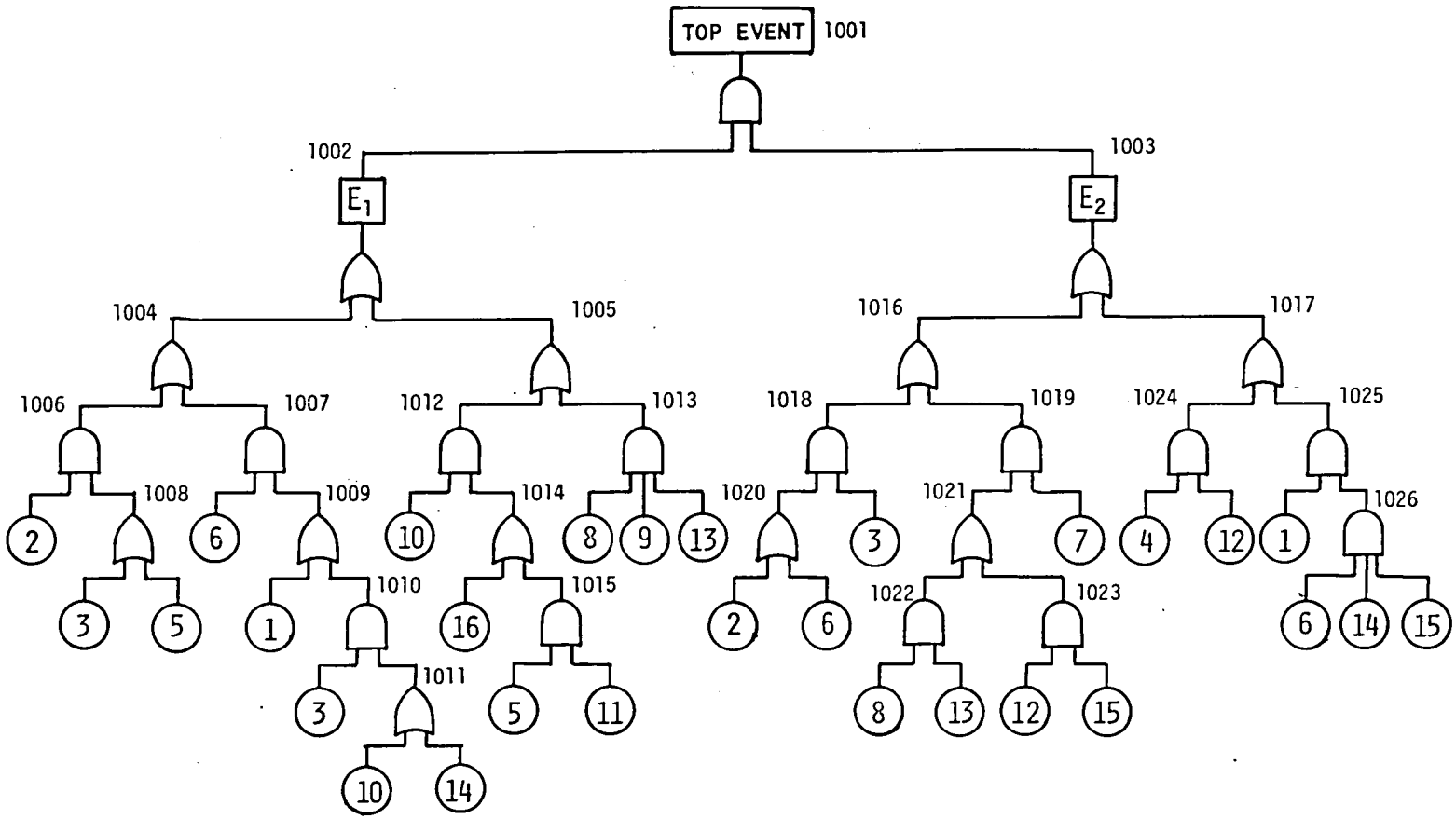


Fig. 23 Illustrative Example of Fault Tree

4.3 $c_{18} \cup c_{24} = \{1,6,7,8,13\} \in C$, (5). $c_{17} \cup c_{21} = \{3,6,8,9,13\} \in C$, (6). $c_{11} \cup c_{21} = \{2,3,5,6\} \supset c_{14}$ and $c_{11} \cup c_{21} \supset c_{14} \cup c_{21}$, (2).

4.4 $c_{11} \cup c_{24} = \{2,5,7,8,13\} \in C$, (8). $c_{17} \cup c_{24} = \{7,8,9,13\} \in C$, (8).

4.5 $c_{13} \cup c_{24} = \{7,8,10,13,16\} \in C$, (3). $c_{16} \cup c_{24} = \{5,7,8,10,11,13\} \in C$, (3). $c_{18} \cup c_{22} = \{1,4,6,12\} \in C$, (2). $c_{18} \cup c_{25} = \{1,6,7,12,15\} \in C$, (2). $c_{11} \cup c_{22} = \{2,4,5,12\} \in C$, (2). $c_{11} \cup c_{25} = \{2,5,7,12,15\} \in C$, (2). $c_{17} \cup c_{22} = \{4,8,9,12,13\} \in C$, (2).

$c_{13} \cup c_{21} = \{3,6,10,16\} \supset c_{12}$, (2). $c_{16} \cup c_{21} = \{3,5,6,10,11\} \supset c_{12}$, (2). $c_{17} \cup c_{25} = \{7,8,9,12,13,15\} \supset c_{24}$, (2).

4.6 Let $c_{13} \cup c_{22} = \{4,10,12,16\} \in C$, $c_{13} \cup c_{25} = \{7,10,12,15,16\} \in C$, $c_{16} \cup c_{22} = \{4,5,10,11,12\} \in C$, $c_{16} \cup c_{25} = \{5,7,10,11,12,15\} \in C$.

Thus C has 20 elements. The total number of checks in Step 4 is 54. Steps 1 & 2 are executed by bit by bit AND/OR operations among binary forms having 16 bits, each corresponding to a primary event, by using bit manipulation technique [71]. The number of AND/OR operations is 13 for Step 1, 14 for Step 2. Step 3 requires 50 additional checks to obtain F, F_{1a}, F_{2a}, F_{1b}, F_{2b}. Thus our algorithm requires 104 checks and 27 AND/OR operations. For comparison, Bennetts' algorithm [8] requires 534 checks without preordering elements of $C_1 \otimes C_2$ into an ascending order of number of primary events of each element, and requires 221 checks even if the preordering is performed (the additional checks for preordering are

not counted). From these facts, the efficiency of our algorithm is apparent.

3.5 Computer Program and Computational Results

We have constructed computer program BUP-CUTS (Bottom-UP algorithm for enumerating minimal CUT Sets of fault tree) in FORTRAN language. The program uses the algorithm presented in Section 3.3. BUP-CUTS program is fully given in [55]. The input and output for the example of Fig. 10 are shown in Fig. 11. The list of top event or each intermediate event must be given as data input along with the number of sons, members of sons (listed in order from the leftmost son) and the logic gate, where primary events are represented by numbers 1, 2, 3, ... (less than 1000), top event by 1001, intermediate events by numbers more than 1001, AND gate by -2 and OR gate by -3. If mutually exclusive primary events appear in the fault tree, then the list of their groups must be also given as data input.

For comparison, we have also constructed another computer program BEN-CUTS which follows Bennetts' algorithm [8] wholly. Both programs use bit manipulation technique [71] to reduce computation time and storage requirements.

Both programs have been tested on a FACOM 230-38; the processing speed of this computer is several 10's of order slower than large digital computers (e.g., FACOM M-190). In Table 4, the computational results of BUP-CUTS program are given along with the results of BEN-CUTS program. Table 4 apparently indicates that BUP-CUTS is efficient and the efficiency becomes noticeable with decreasing number of repeated primary events (which means primary events appearing more than once in the fault tree).

Our algorithm is not compared with Worrell's algorithm [72] and conventional top-down algorithms [17, 61] for the following reasons.

```
**********INPUT**********

UPPER LIMIT TO NUMBER OF MINIMAL CUTSETS =  3000
NUMBER OF PRIMARY EVENTS =    16
NUMBER OF TOP EVENT AND INTERMEDIATE EVENTS =    26
EXISTENCE OF MUTUALLY EXCLUSIVE EVENTS =     0

LIST OF TOP EVENT AND INTERMEDIATE EVENTS

 1001 ( 2) = 1002*1003*  -2*
 1002 ( 2) = 1004*1005*  -3*
 1003 ( 2) = 1016*1017*  -3*
 1004 ( 2) = 1006*1007*  -3*
 1005 ( 2) = 1012*1013*  -3*
 1006 ( 2) =    2*1008*  -2*
 1007 ( 2) =    6*1009*  -2*
 1008 ( 2) =    3*   5*  -3*
 1009 ( 2) =    1*1010*  -3*
 1010 ( 2) =    3*1011*  -2*
 1011 ( 2) =   10*  14*  -3*
 1012 ( 2) =   10*1014*  -2*
 1013 ( 3) =    8*   9*  13*  -2*
 1014 ( 2) =   16*1015*  -3*
 1015 ( 2) =    5*  11*  -2*
 1016 ( 2) = 1018*1019*  -3*
 1017 ( 2) = 1024*1025*  -3*
 1018 ( 2) = 1020*   3*  -2*
 1019 ( 2) = 1021*   7*  -2*
 1020 ( 2) =    2*   6*  -3*
 1021 ( 2) = 1022*1023*  -3*
 1022 ( 2) =    8*  13*  -2*
 1023 ( 2) =   12*  15*  -2*
 1024 ( 2) =    4*  12*  -2*
 1025 ( 2) =    1*1026*  -2*
 1026 ( 3) =    6*  14*  15*  -2*

**********OUTPUT**********

TOTAL NUMBER OF MINIMAL CUTSETS =    20

LIST OF MINIMAL CUTSETS

*  1 *  2  3
*  2 *  1  6 14 15
*  3 *  3  6 10
*  4 *  3  6 14
*  5 *  2  5  7  8 13
*  6 *  2  5  7 12 15
*  7 *  2  4  5 12
*  8 *  1  3  6
*  9 *  1  6  7  8 13
* 10 *  1  6  7 12 15
* 11 *  1  4  6 12
* 12 *  7  8 10 13 16
* 13 *  7 10 12 15 16
* 14 *  4 10 12 16
* 15 *  5  7  8 10 11 13
* 16 *  5  7 10 11 12 15
* 17 *  4  5 10 11 12
* 18 *  3  6  8  9 13
* 19 *  7  8  9 13
* 20 *  4  8  9 12 13
```

Fig. 11

Computer Input and Output for the Example of Fig. 10, Using BUP-CUTS

Table 4

Comparison of Computation Times between BUP-CUTS and BEN-CUTS

Example Number	Number of Primary Events	Number of Repeated Primary Events	Number of Minimal Cut Sets	Computation Time (sec)	
				BUP-CUTS	BEN-CUTS
1	16	0	256	0.5	11.7
2	16	4	256	1.9	14.0
3	16	8	240	0.8	10.7
4	16	16	90	17.3	26.9
5	20	20	357	39.7	150
6	30	15	134	2.6	9.7
7	51	5	457	4.5	118

i. Worrell's algorithm is based on the same principle as Bennetts' algorithm.

ii. At each intermediate stage of implementation of algorithm, a top-down algorithm has a s.o.p. form containing not only primary events but also intermediate events. Accordingly, the combined treatment of primary events and intermediate events is required at each intermediate stage. Hence, a top-down algorithm is more disadvantageous for saving computation time and storage requirement than a bottom-up algorithm that has a s.o.p. form containing only primary events at any stage. Rosenthal [62] has given similar discussions lately.

Chapter 4

A Method for Probabilistic Evaluation of Fault Tree

This chapter presents a method for executing probabilistic evaluation of fault tree efficiently through the combined use of reverse Polish sequence and tree sequence. The method is based on the strategy of first obtaining the symbolic form of top event probability and then computing the top event probability from that symbolic form. Storing the symbolic form makes easy repeated computations of top event probability and sensitivity analysis. This method can extendedly be applied for fault tree containing mutually exclusive primary events.

4.1 Introduction

On executing probabilistic evaluation of fault tree by the Boolean approach, the main efforts have been made to generate a minimal or near-minimal sum-of-product form of the Boolean function relating the top event to primary events, and then, to implement computations by applying successively the addition rule in probability theory [69] or by generating an equivalent disjoint sum-of-product form [8, 16]. For a large scale fault tree, however, the

minimal or near-minimal sum-of-product form is so lengthy that one has been obliged to settle for approximate results, even though a list processing technique is used following Bennetts [8]. Koen and Carnino [32] have analyzed binary fault trees by repeating the operations of identifying the pattern of subsequence contained in the reverse Polish sequence and replacing the pattern with an element. But this method is not effective to the analysis of fault tree containing repeated primary events since the number of patterns becomes very large.

The method presented in this chapter aims to give the exact value of the probability of occurrence of the top event (top event probability), providing an efficient algorithm that implement computations through the combined use of reverse Polish sequence and tree sequence.

This method is based on the strategy of first obtaining the symbolic form of top event probability and then computing the top event probability from that symbolic form. Storing the symbolic form makes easy repeated computations of top event probability and sensitivity analysis. The feature of the method is to repeat recursively two stages, i.e., partitioning fault tree into mutually independent fault trees and reducing fault tree by applying Bayes' theorem, and to reduce the problem to the computations for simple fault trees. The method can extendedly be applied for fault tree containing mutually exclusive primary events.

Section 4.3 provides the method for obtaining the reverse Polish sequence and tree sequence of the symbolic form of top event probability from the sequences of top event. Section 4.4 presents the algorithm for computing the top event probability from the symbolic form. Section 4.5 considers the sensitivity analysis and the treatment of mutually exclusive primary events. Section 4.6 demonstrates the computational results for the containment spray injection system of PWR nuclear power plant [68] obtained by computer program ALFAT of this method.

4.2 Preliminaries

4.2.1 Assumptions and Definitions

In this chapter, the following assumptions on the model of fault tree are made:

1. Only OR, AND and k-out-of-n (k/n) gates are allowed ($k \neq 1, n$). If logic gates, such as NOT, XOR, appear in the fault tree, then it can be transformed into an equivalent fault tree containing only OR, AND and k/n gates by using inversion operations by de Morgan's theorem.

2. In Sections 4.3 & 4.4, we assume that the fault tree contains only independent primary events and no mutually exclusive primary events. This assumptions are relaxed in Section 4.5.

3. The Boolean structure function for the top event need not be coherent.

Since basic terms on fault tree are defined in Chapter 2, only other terms are defined here.

A *repeated primary event* is defined as a primary event that appears more than once in the fault tree. Each repeated primary event is classified into a generation: if a repeated primary events appears in two or more fault subtrees of one gth generation intermediate event and does not appear anywhere else, it is called a *gth-order repeated primary event*.

For the fault tree of Fig. 12, event 2 is a first-order repeated primary event since event 2 appears in two subtrees of top event 1001 (first generation event) which are derived from events 1002 and 1003. Events 8 and 9 are second-order repeated primary events since events 8 and 9 appear in two subtrees of event 1005 (second generation event) which are derived from events 1008 and 1009.

A *simple fault tree* [41] is a fault tree that contains no repeated primary events.

The symbol F is used for the fault tree which is attended to at each stage of this method. Let T be the top event of F and Δ be the logic gate of T. T (or E_i) is also used as binary variable having the value 1 if the top event T (or event i) has occurred and 0 otherwise.

Two or more fault subtrees of T are called to be *mutually dependent* if they are mutually related through one or more first-order repeated primary events; e.g., three fault subtrees FS_1, FS_2, FS_3 of T are mutually dependent if FS_1, FS_2 contain a first-order repeated primary event and FS_2, FS_3 contain another one.

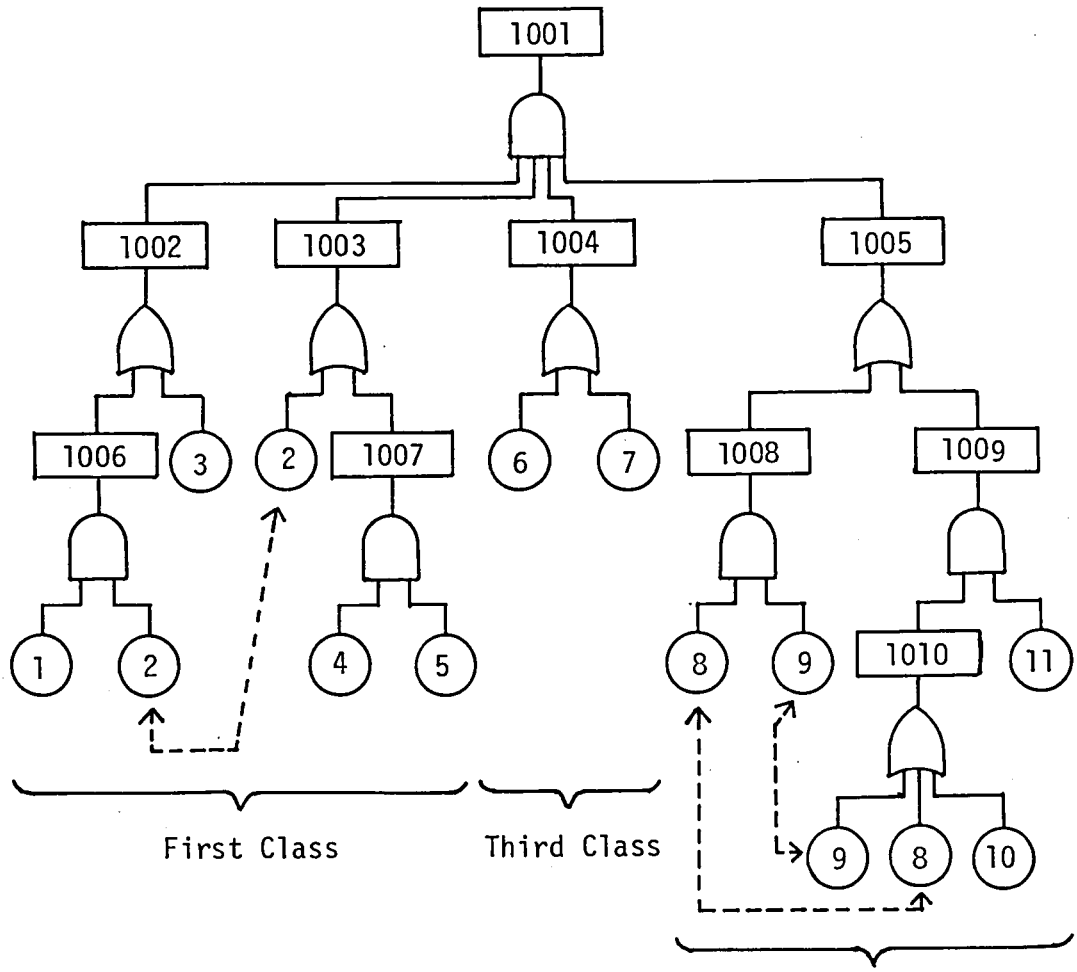


Fig. 12 Example 1 of Fault Tree

Fault tree F can be partitioned into fault trees of the following three classes.

First class: If Δ is AND or OR gate, then the fault tree obtained by joining mutually dependent fault subtrees of T at gate Δ is of the first class. If Δ is k/n gate and F contains one or more first-order repeated primary events, then the whole of F is of the first class. For the fault tree of Fig. 12, the fault tree obtained by joining at AND gate two fault subtrees of top event 1001 that are developed below intermediate events 1002 and 1003 is of the first class, since those fault subtrees are mutually dependent, i.e., they are mutually related through the first-order repeated primary event 2.

Second class: A second class fault tree is a fault subtree of T that contains one or more repeated primary events in itself but no first-order repeated primary events of F (excluding the case where Δ is k/n gate and F is of the first class). For the fault tree of Fig. 12, the fault subtree of top event that is developed below event 1005 is of the second class, since it contains the second-order repeated primary events 8, 9.

Third class: A third class fault tree is a fault subtree of T that contains no repeated primary events of F. This fault tree is a simple fault tree. For the fault tree of Fig.12, the fault subtree of top event that is developed below event 1004 is of the third class.

4.2.2 Symbolic Form of Top Event Probability

The top event probability is expressed as the function of the probabilities p_i of occurrence of primary events i, $i = 1, 2, \ldots$. The *symbolic form* of top event probability is a computer-oriented expression of this function; it is represented by using primary event numbers and basic operators of five kinds, viz., arithmetic sum +, difference - & product x, logical sum V and k/n logic operators. In the computer processing, the reverse Polish sequence

and tree sequence of the symbolic form are used; in the reverse Polish sequence, arithmetic sum, difference & product may be expressed by integers 0, -1, -2, respectively, and logical sum & k/n logic operators by the same integers as Table 2, as shown in Table 5. The *direct form* of top event probability is defined as the form that is expressed by using primary event probabilities p_i and only arithmetic operators +, -, X.

Table 5 is the comparative table of the symbolic form revese Polish sequence & tree sequence and the direct form for the probability of occurrence of event i obtained by performing basic operation on events $i_1, i_2, \ldots, i_n$.

The symbolic form of top event probability is a combination of basic operations shown in Table 5. For example, consider the symbolic form for top event T.

$$T = (1.0 - 2) \times (1 \vee 3 \vee 4) + 2 \times 2/3(1 \times 5, 4, 6) \qquad (4.1)$$

This symbolic form is equivalent to the tree structure composed of the gates of basic operators such as shown in Fig.13 [31].

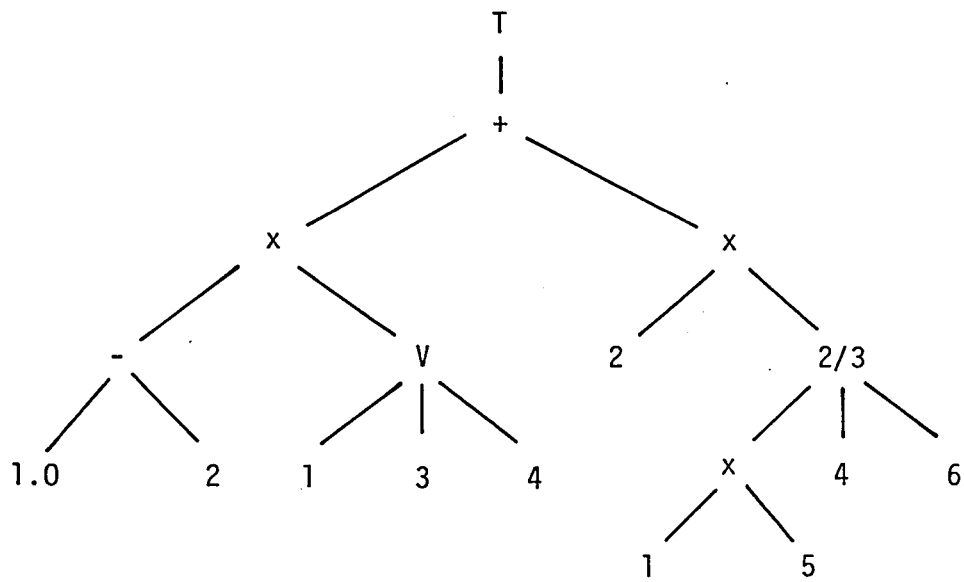


Fig. 13 Tree Structure Equivalent to Symbolic Form (4.1)

Table 5 Symbolic Form and Direct Form for Basic Operations

Basic Operation	Symbolic Form	Reverse Polish Sequence PS(i) & Tree Sequence TSS(i)	Direct Form
Arithmetic Sum	$i = i_1 + i_2 + \ldots + i_n$	$PS(i) = i_1\ i_2\ \ldots\ i_n\ \overset{(0)}{+}$ $TSS(i) = 1\ \ 1\ \ \ldots\ 1\ -(n-1)$	$p_i = p_{i_1} + p_{i_2} + \ldots + p_{i_n}$
Arithmetic Difference	$i = i_1 - i_2$	$PS(i) = i_1\ i_2\ \overset{(-1)}{-}$ $TSS(i) = 1\ \ 1\ \ -1$	$p_i = p_{i_1} - p_{i_2}$
Arithmetic Product	$i = i_1 \times i_2 \times \ldots \times i_n$	$PS(i) = i_1\ i_2\ \ldots\ i_n\ \overset{(-2)}{\times}$ $TSS(i) = 1\ \ 1\ \ \ldots\ 1\ -(n-1)$	$p_i = p_{i_1} p_{i_2} \ldots p_{i_n}$
Logical Sum	$i = i_1 \vee i_2 \vee \ldots \vee i_n$	$PS(i) = i_1\ i_2\ \ldots\ i_n\ \overset{(-3)}{\vee}$ $TSS(i) = 1\ \ 1\ \ \ldots\ 1\ -(n-1)$	$p_i = 1 - (1-p_{i_1})(1-p_{i_2}) \ldots (1-p_{i_n})$
k/n Logic Operation	$i = k/n(i_1, i_2, \ldots, i_n)$	$PS(i) = i_1\ i_2\ \ldots\ i_n\ \overset{(-m)}{k/n}$ $TSS(i) = 1\ \ 1\ \ \ldots\ 1\ -(n-1)$ where $m = \frac{n^2 - 5n + 8}{2} + k + 1$	$p_i = \sum_{j=k}^{n} \sum p_{z_1} p_{z_2} \ldots p_{z_j} \times (1-p_{z_{j+1}}) \ldots (1-p_{z_n})$ where $\{z_1, z_2, \ldots, z_n\}$ is a permutation of $\{1, 2, \ldots, n\}$ and the second summation is taken over all combinations $\{z_1, z_2, \ldots, z_k\}$ of k events out of $\{1, 2, \ldots, n\}$.

The reverse Polish sequence PS(T) and tree sequence TSS(T) of symbolic form (4.1) are obtained by applying the successive substitutions similar to Section 2.3 to the tree structure of Fig.13.

$$PS(T) = 1000\ 2\ -\ 1\ 3\ 4\ \vee\ \times\ 2\ 1\ 5\ \times\ 4\ 6\ 2/3\ \times\ +$$

$$TSS(T) = \quad 1\ \ 1\ -1\ 1\ 1\ 1\ -2\ -1\ 1\ 1\ 1\ -1\ 1\ 1\ -2\ -1\ -1$$

In the reverse Polish sequence, the value 1.0 is represented by 1000.

The direct form of top event probability p_T is:

$$p_T = (1 - p_2) \times \{1 - (1 - p_1)(1 - p_3)(1 - p_4)\} + p_2 \times \{(1 - p_1p_5)p_4p_6$$

$$+ p_1p_5(1 - p_4)p_6 + p_1p_5p_4(1 - p_6) + p_1p_5p_4p_6\}$$

The feature of the symbolic form exists in using logical sum and k/n logic operators without transforming them into equivalent arithmetic operations. Storing the symbolic form in a computer is useful for repeated computations of top event probabilities and saves memories since the symbolic form is shorter than the direct form. The top event probability can be computed through the last-in first-out push-down stack processing using the reverse Polish sequence and tree sequence of symbolic form, as shown in Section 4.4.

4.3 Method for Obtaining Symbolic Form of Top Event Probability

In this section a method is presented for obtaining the sequence representation of symbolic form of top event probability from the sequence representation of top event defined in Chapter 2. This method is based on the principle of repeating partition and reduction of fault tree recursively until reaching symbolic forms of simple fault trees. In what follows, three basic stages of this

method, i.e., partition, reduction & symbolic form of simple fault tree, and the overall algorithm are explained. In Sections 4.3 & 4.4, logic gates in reverse Polish sequences are expressed by integers such as shown in Tables 2 & 5, with a mind to using a computer.

4.3.1 Basic Stages of Method

Stage 1: Partition of Fault Tree

As shown in 4.2.1, fault tree F can be partitioned into fault trees each of which belongs to some of three classes.

Let F_r (r = 1, 2, ..., L; $L \leq l$, where l is the number of sons of T) be the fault trees obtained by partioning F, and T_r be the top event of F_r. Fault trees F_r are mutually independent, and the fault tree obtained by joining all fault trees F_r at logic gate Δ is equivalent to F. (If L = 1, then F_1 is equal to F.)

If Δ is AND gate, then

$$\Pr\{T = 1\} = \Pi_{r=1}^{L} \Pr\{T_r = 1\}. \tag{4.2}$$

The symbolic form of (4.2) is

$$T = T_1 \times T_2 \times ..... \times T_L. \tag{4.3}$$

PS(T) and TSS(T) are represented by the following relations.

$$\begin{aligned} PS(T) &= PS(T_1)\ PS(T_2)\ .....\ PS(T_L)\ -2 \\ TSS(T) &= TSS(T_1)\ TSS(T_2)\ .....\ TSS(T_L)\ -(L-1) \end{aligned} \tag{4.4}$$

If Δ is OR gate, then

$$\Pr\{T = 1\} = 1 - \Pi_{r=1}^{L} (1 - \Pr\{T_r = 1\}). \tag{4.5}$$

The symbolic form of (4.5) is

$$T = T_1 \vee T_2 \vee ..... \vee T_L. \tag{4.6}$$

Then PS(T) is

$$PS(T) = PS(T_1)\ PS(T_2)\ \ldots\ PS(T_L)\ -3. \tag{4.7}$$

TSS(T) is same as (4.4).

If Δ is k/L gate, then

$$\Pr\{T = 1\} = \sum_{\xi=k}^{L} \sum \Pr\{T_{z_1} = 1\} \Pr\{T_{z_2} = 1\} \ldots \Pr\{T_{z_\xi} = 1\} \times (1 - \Pr\{T_{z_{\xi+1}} = 1\}) \ldots (1 - \Pr\{T_{z_L} = 1\}), \tag{4.8}$$

where $\{z_1, z_2, \ldots, z_\xi, z_{\xi+1}, \ldots, z_L\}$ is a permutation of $\{1, 2, \ldots, L\}$ and the second summation is taken over all combinations $\{z_1, z_2, \ldots, z_\xi\}$ of ξ events out of $\{1, 2, \ldots, L\}$.
The symbolic form of (4.8) is

$$T = k/L(T_1, T_2, \ldots, T_L). \tag{4.9}$$

Then PS(T) is

$$PS(T) = PS(T_1)\ PS(T_2)\ \ldots\ PS(T_L)\ -m, \tag{4.10}$$

where $-m$ is the representation of k/L gate (See Tables 2, 3 of Chapter 2). TSS(T) is same as (4.4).

Accordingly the problem of computing $\Pr\{T = 1\}$ is decomposed into the problems of computing $\Pr\{T_r = 1\}$, $r = 1, 2, \ldots, L$.

The partition of F is implemented by identifying the sons of tree sequence TS(T) by algorithm A.2 presented in Section 2.5 and then finding repeated primary events in the subsequences of reverse Polish sequence P(T) corresponding to those sons.

Stage 2: Reduction of Fault Tree

When Bayes' theorem is applied to a first-order repeated primary event i of a first-class fault tree F, we obtain

$$\Pr\{T = 1\} = \Pr\{T = 1/E_i = 0\}\times(1 - p_i) + \Pr\{T = 1/E_i = 1\}\times p_i. \tag{4.11}$$

Accordingly the problem of computing $\Pr\{T = 1\}$ is decomposed into the problems of computing top event probabilities of two fault trees obtained by setting $E_i = 0$ and $E_i = 1$, respectively, and reducing F by using the identities $1 \vee X = 1$, $1 \wedge X = X$, $0 \vee X = X$, $0 \wedge X = 0$, $k/n(1, a_1, a_2, \ldots, a_{n-1}) = (k-1)/(n-1)(a_1, a_2, \ldots, a_{n-1})$ (if $k = 2$, then the right hand side is equal to $a_1 \vee a_2 \vee \ldots \vee a_{n-1}$), $k/n\,(0, a_1, a_2, \ldots, a_{n-1}) = k/(n-1)(a_1, a_2, \ldots, a_{n-1})$ (if $k = n - 1$, then the right hand side is equal to $a_1 \wedge a_2 \wedge \cdots \wedge a_{n-1}$).

Let T_0, T_1 be the top events of two resulting fault trees. Then, the symbolic form of (4.11) is

$$T = T_0 \times (1.0 - i) + T_1 \times i. \tag{4.12}$$

PS(T) and TSS(T) are represented by the following relations.

$$\begin{aligned} \mathrm{PS}(T) &= \mathrm{PS}(T_0)\ 1000\ i\ -1\ -2\ \ \mathrm{PS}(T_1)\ i\ -2\ \ 0 \\ \mathrm{TSS}(T) &= \mathrm{TSS}(T_0)\ \ \ 1\ \ \ 1\ -1\ -1\ \mathrm{TSS}(T_1)\ 1\ -1\ -1 \end{aligned} \tag{4.13}$$

The algorithm is presented below for implementing this reduction process by using reverse Polish sequence P(T) and tree sequence TS(T).

(AL.1) Algorithm for Reducing P(T) and TS(T) with Respect to Primay Event i:

Let E be the element of TS(T) corresponding to event i in P(T). Find the ascending path from E by algorithm A.3 in Section 2.5, and then let AS(i) be the sequence of elements of P(T) corresponding to this ascending path.

1) If AS(i) = i $-m$ ..... ($m \geq 4$):

As shown in Tables 2 & 3 in Section 2.3, element $-m$ means k/n gate; k and n are obtained from m as follows.

$$n = 2 + \left[\frac{\sqrt{8(m-4)+1}}{2} + \frac{1}{2} \right] \tag{4.14}$$

$$k = m - \frac{n^2 - 5n + 8}{2} - 1 \tag{4.15}$$

$[\,x\,]$ is the maximum integer not greater than x. Eqs. (4.14) and (4.15) are proved in Appendix D.

Let $-\gamma$ be the element of TS(T) corresponding to $-m$.

When setting $E_i = 1$: Change $-\gamma$ into $-(\gamma-1)$ in TS(T). If $k > 2$, then change $-m$ into $-(m-n+2)$ in P(T), where element $-(m-n+2)$ means $(k\text{-}1)/(n\text{-}1)$ gate. If $k = 2$, then change $-m$ into -3 (OR gate) in P(T).

When setting $E_i = 0$: Change $-\gamma$ into $-(\gamma-1)$ in TS(T). If $k \neq n-1$, then change $-m$ into $-(m-n+3)$ in P(T), where element $-(m-n+3)$ means $k/(n\text{-}1)$ gate. If $k = n-1$, then change $-m$ into -2 (AND gate) in P(T).

2) If AS(i) = i -3 -3 ... -3 $-m$ .... ($m = 2$ or $m \geq 4$):

Let ν_i be the number of OR gates (-3) which appear in succession after i in AS(i), and let G_a, G_b be the first and last of ν_i OR gates, respectively. Let $-\alpha$, $-\beta$, $-\gamma$ be the elements of TS(T) corresponding to G_a, G_b, $-m$, respectively.

When setting $E_i = 1$: Remove the tree subsequence whose representative element is $-\beta$ and the corresponding subsequence of P(T) from TS(T) and P(T), respectively. If $\gamma = 1$, then remove $-\gamma$ in TS(T) and $-m$ in P(T). If $\gamma \geq 2$, then change $-\gamma$ into $-(\gamma-1)$ in TS(T). If $m \geq 4$, then compute n, k by Eqs. (4.14), (4.15); change $-m$ into $-(m-n+2)$ in P(T) if $k > 2$, and change $-m$ into -3 in P(T) if $k = 2$. If i is followed, in AS(i), by only OR gates without appearance of $-m$, then remove P(T), TS(T) and set $T = 1$.

When setting $E_i = 0$: Remove event i in P(T) and the corresponding element in TS(T). If $\alpha \geq 2$, then change $-\alpha$ into $-(\alpha-1)$ in TS(T). If $\alpha = 1$, then remove $-\alpha$ in TS(T) and G_a in P(T).

3) If AS(i) = i -2 -2 ... -2 -m .... ($m \geq 3$):

The algorithm is same as 2), on condition that it is read by changing terms AND gate, OR gate, $E_i = 0$, $E_i = 1$, $T = 1$ into OR gate, AND gate, $E_i = 1$, $E_i = 0$, $T = 0$, respectively and changing the underlined sentence into "change -m into -($m - n + 3$) in P(T) if $k \neq n - 1$, and change -m into -2 in P(T) if $k = n - 1$".

When M first-order repeated primary events ($M \geq 2$) are contained in F, the process of applying AP-1 successively to all those events yields 2^M *reduced fault trees*, i.e., fault trees free of first-order repeated primary events. Hence the number of resulting reduced fault trees extremely increases with M. The following algorithm reduces the number of resulting reduced fault trees by appropriately choosing the order of applying AL-1 to first-order repeated primary events.

(AL-2) Algorithm for Obtaining Reduced Fault Trees from F:

Step 1: Put F as F_c, then set $p \leftarrow 1$, $q \leftarrow 1$, and go to Step 2.

Step 2: If F_c contains two or more first-order repeated primary events, then go to Step 3.

If F_c contains only one first-order repeated primary event, then put that event as s_p and go to Step 4.

If F_c contains no first-order repeated primary event, then F_c is qth reduced fault tree of F; then set $q \leftarrow q + 1$, $p \leftarrow p - 1$, and go to Step 5.

Step 3: Let e_μ, $\mu = 1, 2, \ldots, \kappa$ be the first-order repeated primary events of F_c. For each event e_μ, let $h_1(\mu)$ (or $h_0(\mu)$) be the number of first-order repeated primary events of e_μ which are changed into non-first-order by setting $E_{e_\mu} = 1$ (or $E_{e_\mu} = 0$) and implementing the reduction by AL-1. Then choose from $\{e_1, e_2, \ldots, e_\kappa\}$ the event that has the maximum value of $h_1(\mu) + h_0(\mu)$. Put that event as s_p and go to Step 4.

Step 4: Set $V_p = 0$, $E_{s_p} = 0$ and implement the reduction of F_c by AL-1. Reset the resulting fault tree as F_c, and go to Step 2.

Step 5: If $V_p = 0$, then reset $V_p = 1$. Set $E_{s_1} = V_1$, $E_{s_2} = V_2$, ..., $E_{s_p} = V_p$ and implement the reduction of F by applying AP-1 repeatedly. Reset the resulting fault tree as F_c, then set $p \leftarrow p + 1$ and go to Step 2.

If $V_p = 1$ and $p \neq 1$, then set $p \leftarrow p - 1$ and repeat Step 5.

If $V_p = 1$ and $p = 1$, then stop.

Stage 3: Symbolic Form of Top Event Probability of Simple Fault Tree

For a simple fault tree F, reverse Polish sequence P(T) and tree sequence TS(T) of top event are diverted as PS(T) and TSS(T) of the symbolic form of top event probability, respectively, for the following reasons.

The inputs of each logic gate are mutually independent since F contains no repeated primary events. The top event probability is hence obtained by applying successively from bottom up the probability computation rules for logic gates with independent inputs given in Eqs.(4.2),(4.5),(4.8). Accordingly the symbolic form of top event probability is obtained from the Boolean function for top event T simply by changing each logical product into the arithmetic product. This means that PS(T) and TSS(T) are equal to P(T) and TS(T),respectively, since the arithmetic product is represented by -2 in PS(T).

4.3.2 Overall Algorithm

The overall algorithm for obtaining the symbolic form of top event probability of the original fault tree is as follows:

Step 1: Put the given fault tree as F, then set $u \leftarrow 1$, $r_u \leftarrow 1$, and go to Step 2.

Step 2: Implement the partition of F by Stage 1. Put the resulting fault trees F_r as $F(u, r)$, $r = 1, 2, ..., L$. Set $L_u \leftarrow L$,

and go to Step 3.

Step 3: Put $F(u, r_u)$ as F. If F is of the first class, then set $q_u \leftarrow 1$ and go to Step 4. If F is of the second class, then set $u \leftarrow u + 1$, $r_u \leftarrow 1$, and go to Step 2. If F is of the third class, then go to Step 5.

Step 4: Generate the q_uth reduced fault tree of F by algorithm AP-2 in Stage 2, and reset it as F. If F is a simple fault tree, then go to Step 5. If not so, then set $u \leftarrow u + 1$, $r_u \leftarrow 1$ and go to Step 2.

Step 5: Put sequences P(T) and TS(T) of top event T of F as sequences PS(T) ans TSS(T) of symbolic form of top event probability, respectively, by Stage 3, and go to Step 6.

Step 6: Put $F(u, r_u)$ as F. If there exist no more reduced fault trees of F, then go to Step 7. If not so, then set $q_u \leftarrow q_u + 1$ and go to Step 4.

Step 7: If $r_u \neq L_u$, then set $r_u \leftarrow r_u + 1$ and go to Step 3. If $r_u = L_u$ and $u \neq 1$, then set $u \leftarrow u - 1$ and go to Step 6. If $r_u = L_u$ and $u = 1$, then obtain the sequence representation of symbolic form of top event probability of original fault tree by substituting the result of each stage into the equivalent event.

This algorithm uses a backtracking technique to implement the method effectively. For the example of Fig. 14, the process of reaching simple fault trees by the method is illustrated in Fig. 15. For this example, the procedure of method is followed corresponding to the above steps.

1. (Step 1-3) Put the fault tree of Fig. 14 as F. Set $u \leftarrow 1$, $r_1 \leftarrow 1$. Since entire F is of the first class, set $L_1 \leftarrow 1$ and $F(1, 1) = F_1 = F$.

2. (Step 4) By Stage 2, the reduced fault trees of $F(1, 1)$ are obtained as follows:

The first-order repeated primary events of $F(1, 1)$ are 3, 4. Event 3 is choosed by Step 3 of AL-2. Implement the reduction of

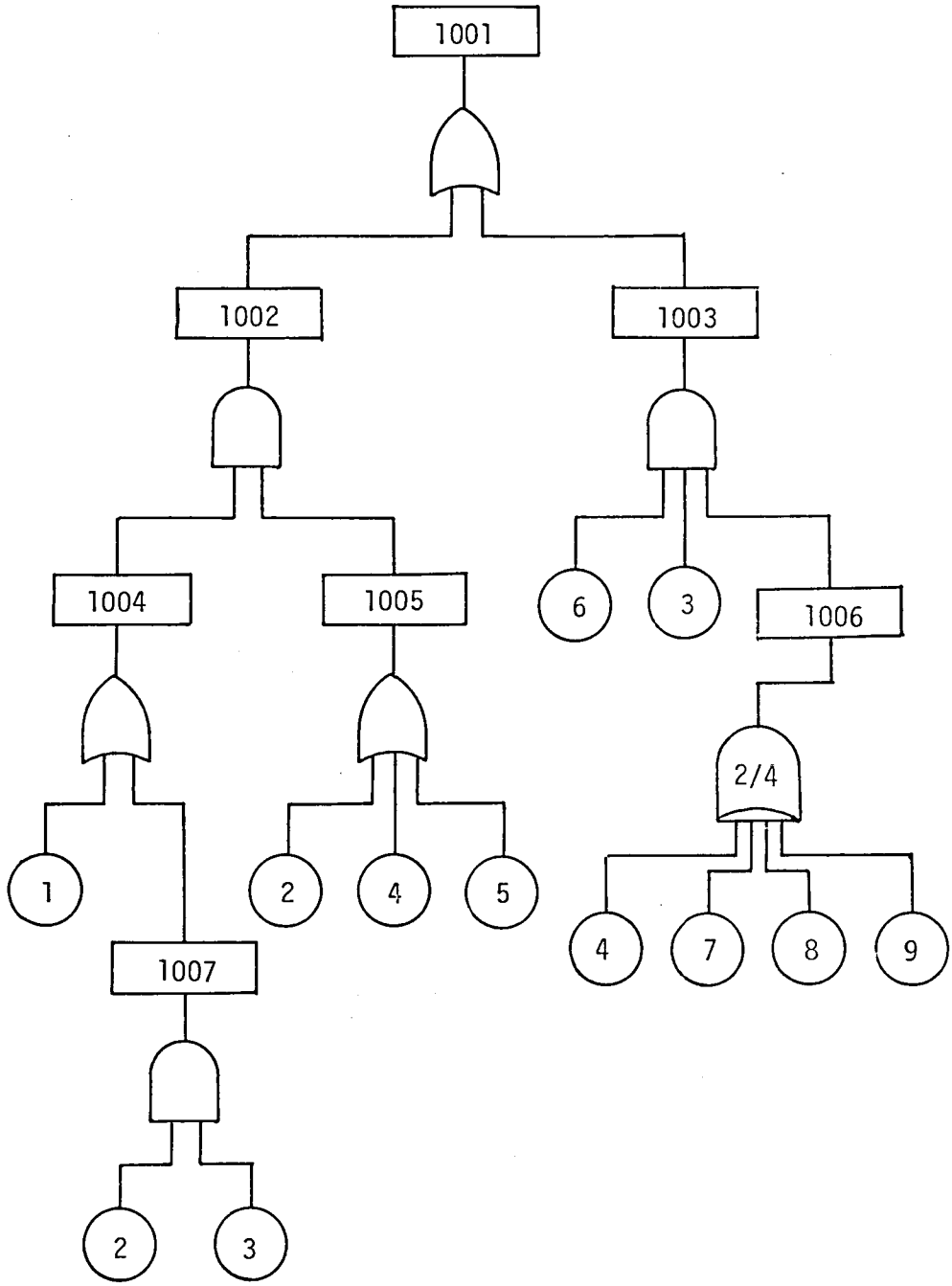


Fig. 14 Example 2 of Fault Tree

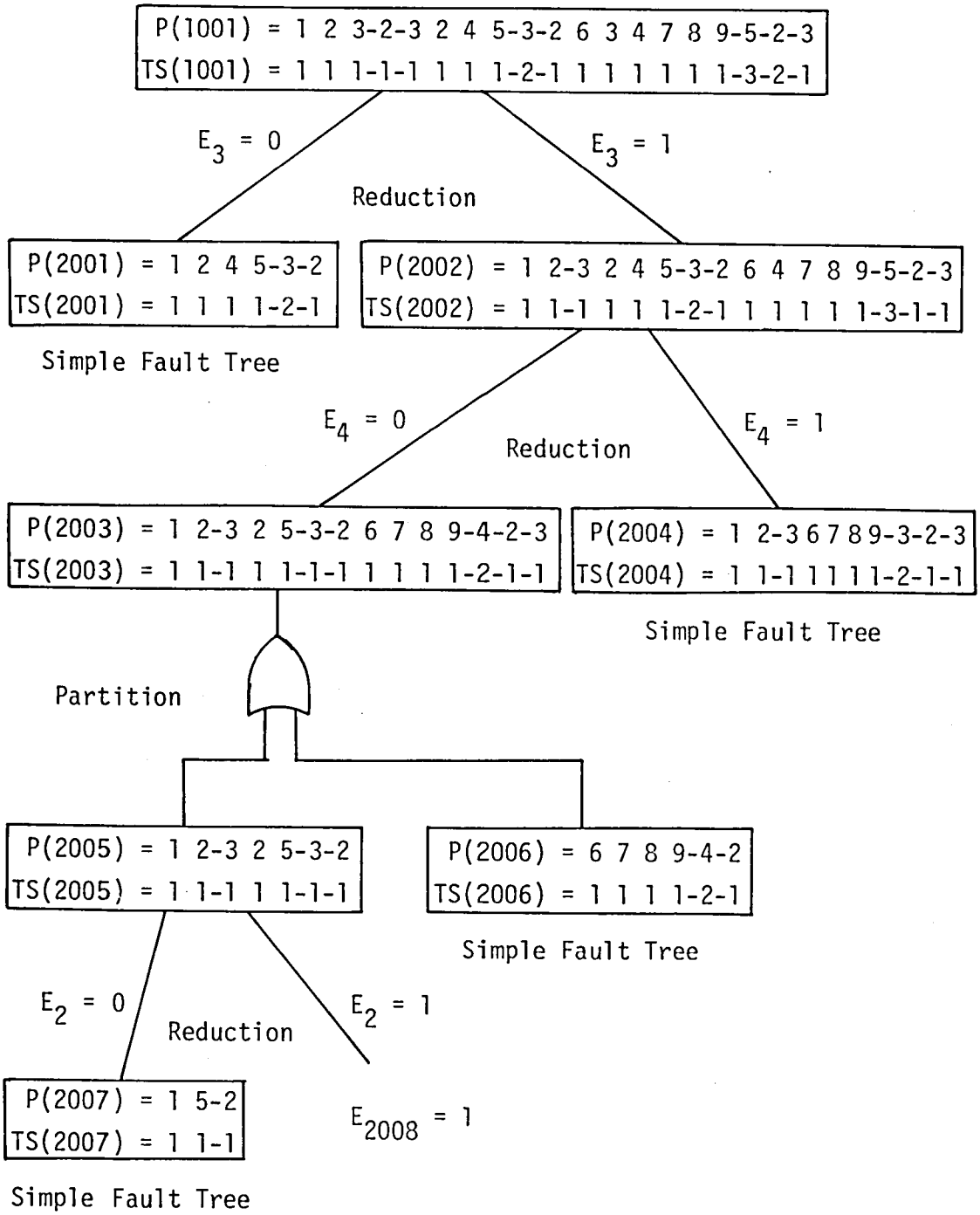


Fig. 15 Procedural Illustration of Method for Example 2

$F(1, 1)$ by setting $E_3 = 0$, $E_3 = 1$, and let 2001, 2002 be the top events of resulting fault trees FT_1, FT_2, respectively. The sequences of each top event are shown in Fig. 15. From Eq. (4.13),

$$\begin{aligned} \text{PS}(1001) &= \text{PS}(2001)\ 1000\ 3\ -1\ -2\ \ \text{PS}(2002)\ 3\ -2\ \ 0 \\ \text{TSS}(1001) &= \text{TSS}(2001)\ \ \ 1\ \ \ 1\ -1\ -1\ \text{TSS}(2002)\ 1\ -1\ -1 \end{aligned} \tag{4.16}$$

Since FT_1 contains no first-order repeated primary event, FT_1 is the first reduced fault tree of $F(1, 1)$. Implement the reduction of FT_2 by setting $E_4 = 0$, $E_4 = 1$, and let 2003, 2004 be the top events of resulting fault trees FT_3, FT_4, respectively. Then,

$$\begin{aligned} \text{PS}(2002) &= \text{PS}(2003)\ 1000\ 4\ -1\ -2\ \ \text{PS}(2004)\ 4\ -2\ \ 0 \\ \text{TSS}(2002) &= \text{TSS}(2003)\ \ \ 1\ \ \ 1\ -1\ -1\ \text{TSS}(2004)\ 1\ -1\ -1 \end{aligned} \tag{4.17}$$

FT_3, FT_4 are the second and third reduced fault tree, respectively.

3. (Step 5) Since FT_1 is simple,

$$\text{PS}(2001) = \text{P}(2001), \quad \text{TSS}(2001) = \text{TS}(2001). \tag{4.18}$$

4. (Step 6, 4, 2) Since FT_3 is not simple, set $u \leftarrow 2$, $r_2 \leftarrow 1$. By the partition of FT_3, two fault trees FT_5, FT_6 (top events 2005, 2006) are obtained. From Eq. (4.7),

$$\begin{aligned} \text{PS}(2003) &= \text{PS}(2005)\ \ \text{PS}(2006)\ -3 \\ \text{TSS}(2003) &= \text{TSS}(2005)\ \text{TSS}(2006)\ -1 \end{aligned} \tag{4.19}$$

Set $L_2 \leftarrow 2$ and $F(2,1) = FT_5$, $F(2, 2) = FT_6$.

5. (Step 3,4) $F(2, 1)$ is of the first class. Event 2 is the first-order repeated primary event. Implement the reduction of $F(2, 1)$ by setting $E_2 = 0$, $E_2 = 1$, and let 2007, 2008 be the top events of resulting fault trees FT_7, FT_8, respectively.

$$\begin{aligned} \text{PS}(2005) &= \text{PS}(2007)\ 1000\ 2\ -1\ -2\ \ \text{PS}(2008)\ 2\ -2\ \ 0 \\ \text{TSS}(2005) &= \text{TSS}(2007)\ \ \ 1\ \ \ 1\ -1\ -1\ \text{TSS}(2008)\ 1\ -1\ -1 \end{aligned} \tag{4.20}$$

FT_7, FT_8 are the first and second reduced fault tree of $F(2, 1)$, respectively.

6. (Step 5) Since FT_7 is simple,

$$PS(2007) = P(2007), \quad TSS(2007) = TS(2007). \tag{4.21}$$

7. (Step 6, 4, 5) Since $E_{2008} = 1$ for FT_8,

$$PS(2008) = 1000, \quad TS(2008) = 1. \tag{4.22}$$

8. (Step 6, 7) Since $r_2 \neq L_2$, set $r_2 \leftarrow 2$.

9. (Step 3, 5) $F(2, 2)$ is of the third class. Therefore,

$$PS(2006) = P(2006), \quad TSS(2006) = TS(2006). \tag{4.23}$$

10. (Step 6, 7) Since $r_2 = L_2$, set $u \leftarrow 1$.

11. (Step 6, 4, 5) The third reduced fault tree of FT_4 is simple. Therefore,

$$PS(2004) = P(2004), \quad TSS(2004) = TS(2004). \tag{4.24}$$

12. (Step 6, 7) $r_1 = L_1$ and $u = 1$. The complete forms of PS(1001) and TSS(1001) are obtained by the successive substitutions using Eqs. (4.16) - (4.24) and the sequences of top events of simple fault trees shown in Fig. 15.

$$\begin{aligned} PS(1001) = {} & 1\ 2\ 4\ 5\ -3\ -2\ 1000\ 3\ -1\ -2\ 1\ 5\ -2\ 1000\ 2\ -1\ -2\ 2\ \ 0 \\ & 6\ 7\ 8\ 9\ -4\ -2\ -3\ 1000\ 4\ -1\ -2\ 1\ 2\ -3\ 6\ 7\ 8\ 9\ -3\ -2\ -3 \\ & 4\ -2\ \ 0\ 3\ -2\ \ 0 \\ TSS(1001) = {} & 1\ 1\ 1\ 1\ -2\ -1\ \ 1\ \ 1\ -1\ -1\ 1\ 1\ -1\ \ 1\ \ 1\ -1\ -1\ 1\ -1 \\ & 1\ 1\ 1\ 1\ -2\ -1\ -1\ \ 1\ \ 1\ -1\ -1\ 1\ 1\ -1\ 1\ 1\ 1\ 1\ -2\ -1\ -1 \\ & 1\ -1\ -1\ 1\ -1\ -1 \end{aligned} \tag{4.25}$$

4.4 Algorithm for Computing Top Event Probability from Symbilic Form

In this section the algorithm is presented for computing the top event probability from the symbolic form obtained by the method of Section 4.3.

Let N be the length of PS(1001) and TSS(1001) for top event 1001 of fault tree. Let G_1, G_2, ..., G_N be the elements of PS(1001) and H_1, H_2, ..., H_N be the elements of TSS(1001) in the order of arranging.

The top event probability is computed by repeating the process of identifying the inputs of each operator in PS(1001) and applying the corresponding operation to the probabilities of occurrence of those inputs. For doing this, it is convenient to use an auxiliary stack, i.e., a last-in first-out pushdown list which is a conventional tool in the list processing programming technique.

Let v be the stack variable of stack St, i.e., the number of elements being pushed down to St.

(AL-3) Algorithm for Computing Top Event Probability:

Step 1: Set $w \leftarrow 1$, $v \leftarrow 0$, and go to Step 2.

Step 2: If $H_w = 1$ and $G_w \neq 1000$, then push down the probability of occurrence of event G_w as $St(v+1)$. If $H_w = 1$ and $G_w = 1000$, then push down value 1.0 as $St(v+1)$. Set $v \leftarrow v + 1$, $w \leftarrow w + 1$, and repeat Step 2.

If H_w is $-\zeta$ ($\zeta > 0$), then pop up $St(v)$, $St(v-1)$, ..., $St(v-\zeta)$. Then, implement the following computation and push down the result as $St(v-\zeta)$:

1) If $G_w = 0$, then compute $\Sigma_{j=0}^{\zeta} St(v-j)$.

2) If $G_w = -1$, then compute $St(v-1) - St(v)$.

3) if $G_w = -2$, then compute $\Pi_{j=0}^{\zeta} St(v-j)$.

4) If $G_w = -3$, then compute $1 - \prod_{j=0}^{\zeta}(1 - St(v-j))$.

5) If $G_w = -m$ $(m \geq 4)$, then implement the probability computation for $k/(\zeta+1)$ logic operation of events with the probabilities of occurrence $St(v)$, $St(v-1)$, ..., $St(v-\zeta)$ by using the method mentioned below, where k is obtained by Eq. (4.15).

Set $v \leftarrow v - \zeta$, and go to Step 3.

Step 3: If $w \neq N$, then set $w \leftarrow w + 1$ and go to Step 2. If $w = N$, then stop; $St(1)$ is the top event probability.

Now consider the case of computing the probability of occurrence of the output of k/n logic gate with n independent inputs 1, 2, ..., n whose probabilities of occurrence are p_1, p_2, ..., p_n, respectively. In this case, it is too time-consuming to use Eq. (4.8) directly. But the computation time can be reduced by using the following method.

Consider the process that a binary tree is constituted such that the branch of level i corresponds to either state of "event i occurs $(E_i = 1)$" and "event i does not occur $(E_i = 0)$". Then,

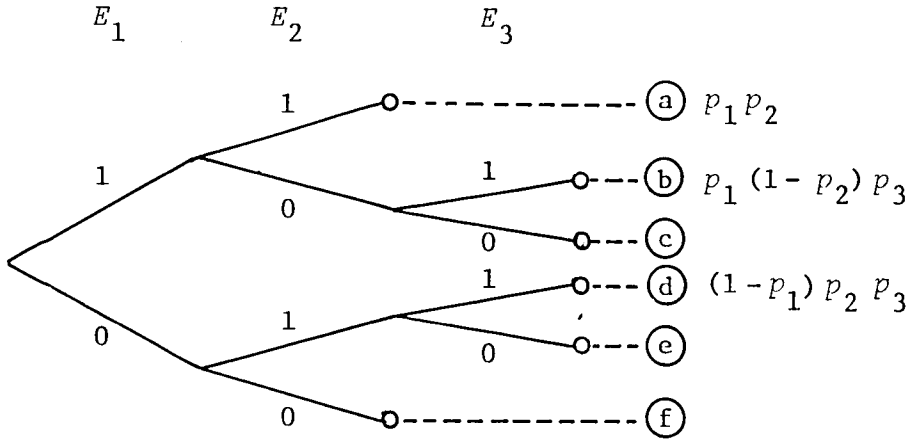


Fig. 16 Probability Computation for 2/3 Logic

i) If the number of inputs having value 1 becomes k, then the output occurs regardless of the states of remaining inputs.

ii) If the number of inputs having value 0 becomes $n-k+1$, then the output does not occur regardless of the states of remaining inputs.

Therefore, when either of the above two conditions is satisfied, it is not necessary to continue more branches. The probability of occurrence of the output is obtained by adding the probabilities of case i).

For 2/3 gate, this method is illustrated in Fig. 16. It is necessary only to compute the probabilities for cases a, b, d where condition i) is satisfied. It is not necessary to continue the branches for remaining inputs for cases c, e, f where condition ii) is satisfied. Therefore, the probability of occurrence of the output is $p_1p_2 + p_1(1 - p_2)p_3 + (1 - p_1)p_2p_3$.

4.5 Application and Extension of Method

4.5.1 Sensitivity Analysis

Needless to say, Eq. (4.11) holds for all primary events. Eq. (4.11) shows that the top event probability is linear in each primary event probability. Therefore, variation Δp in top event probability corresponding to variation Δp_i in probability p_i of event i is given by the following relation [36].

$$\Delta p = s_i \Delta p_i \tag{4.26}$$

where

$$s_i \equiv \Pr\{T = 1/E_i = 1\} - \Pr\{T = 1/E_i = 0\}. \tag{4.27}$$

s_i is called the sensitivity coefficient of event i.

It is derived from Eq. (4.11) that $\Pr\{T = 1/E_i = 1\}$ is the top event probability in the case of $p_i = 1$. Accordingly it is computed by substituting value 1 for p_i in algorithm AL-3 of Section 4.4.

$\Pr\{T = 1/E_i = 0\}$ is computed from $\Pr\{T = 1\}$ and $\Pr\{T = 1/E_i = 0\}$ by using Eq. (4.11).

Varation Δp in top event probability corresponding to small variations Δp_i, $i = 1, 2, \ldots, n_e$ in primary event probabilities is

$$\Delta p \simeq \Sigma_{i=1}^{n_e} s_i \, \Delta p_i. \tag{4.28}$$

Sensitivity coefficients have further applications to fault tree having coherent structure.

The criticality importance of event i [34] is

$$C_i = s_i \, p_i / p_T, \tag{4.29}$$

where p_T is the top event probability.

For time-dependent case, the values of Eqs. (4.26)-(4.27) at time t are computed from values $p_i(t)$, $p_T(t)$, $s_i(t)$ at time t.

Let $W(t_1, t_2)$ be the expected number of occurrence of top event in time interval $[t_1, t_2]$ and $w_i(t)dt$ be the probability that event i is not occurring at time t and occurs in $[t, t+dt]$. Then,

$$W(t_1, t_2) = \Sigma_{i=1}^{n_e} \int_{t_1}^{t_2} s_i(t) \, w_i(t) \, dt. \tag{4.30}$$

In the case of no repair, $W(t_1, t_2)$ is equal to the probability of occurrence of top event in $[t_1, t_2]$.

Barlow-Proschan's importance $BP_i(t)$ of event i [7] is

$$BP_i(t) = \int_0^t s_i(t) \, w_i(t) \, dt \; / \; W(0, t). \tag{4.31}$$

Even if $p_i(t)$ is given as the unavailability of component i, the above discussions hold. Event i could be an intermediate event if it is independent of other events (excluding events of the fault tree which is developed below event i).

Consider the case where we want to investigate how it affects the system failure probability to add a fault tree or primary event as a part of the original fault tree. So is the case of adding a new protection system or some other redundant units to the system.

Suppose that fault tree F_a is added as an input of logic gate G_a of the original fault tree. Let T_a be the top event of F_a and p_a be the probability of occurrence of T_a. F_a is assumed to be independent of events of the original fault tree.

Suppose that the probability of occurrence of top event varies from p_T to p_T' by adding F_a. Then, it is shown that p_T' is obtained from the information about the original fault tree. From Bayes' theorem,

$$p_T' = \Pr\{T = 1/T_a = 0\} \times (1 - p_a) + \Pr\{T = 1/T_a = 1\} \times p_a.$$

If G_a is AND gate, then

$$\Pr\{T = 1/T_a = 1\} = p_T.$$

$\Pr\{T = 1/T_a = 0\}$ is equal to the probability of occurrence of T for the case of setting the output of G_a to 0. If there exists a non-repeated primary event i which is a son of G_a or a descendant of G_a through only AND gates, then

$$\Pr\{T = 1/T_a = 0\} = \Pr\{T = 1/E_i = 0\}.$$

If G_a is OR gate, then the same discussions are made by changing $T_a = 1$, $T_a = 0$, $E_i = 0$, AND gate into $T_a = 0$, $T_a = 1$, $E_i = 1$, OR gate, respectively.

4.5.2 Treatment of Mutually Exclusive Primary Events

Suppose that the fault tree contains a set $\Phi_i \equiv \{(i,1), (i,2), \ldots, (i,n_i)\}$ of mutually exclusive primary events. Let p_{ij} be the probability of occurrence of event (i,j).

If Φ_i is exhaustive, then Bayes' theorem is

$$\Pr\{T = 1\} = \sum_{j=1}^{n_i} \Pr\{T = 1/E_{i1} = 0, \ldots, E_{i,j-1} = 0, E_{ij} = 1, E_{i,j+1} = 0, \ldots, E_{in_i} = 0\}\, p_{ij}, \quad (4.32)$$

where $\sum_{j=1}^{n_i} p_{ij} = 1.$

If Φ_i is not exhaustive, then

$$\Pr\{T = 1\} = \Pr\{T = 1/E_{i1} = 0, E_{i2} = 0, \ldots, E_{in_i} = 0\}\,(1 - \sum_{j=1}^{n_i} p_{ij}) + \sum_{j=1}^{n_i} \Pr\{T = 1/E_{i1} = 0, \ldots, E_{i,j-1} = 0, E_{ij} = 1, E_{i,j+1} = 0, \ldots, E_{in_i} = 0\}\, p_{ij}. \quad (4.33)$$

Our method can extendedly be applied by using Eq.(4.32) or (4.33) instead of Eq. (4.11) in Stage 2. The algorithm would be implemented by using MS(T) or TMS(T) defined in Section 2.6 in addition to P(T) and TS(T).

If Φ_i is not exhaustive, then variation Δp in top event probability corresponding to variations Δp_{ij} of p_{ij}, $j = 1, 2, \ldots, n_i$ is

$$\Delta p = \sum_{j=1}^{n_i} s_{ij}\, \Delta p_{ij}, \quad (4.34)$$

where $s_{ij} \equiv \Pr\{T = 1/E_{i1} = 0, \ldots, E_{i,j-1} = 0, E_{ij} = 1, E_{i,j+1} = 0, \ldots, E_{in_i} = 0\} - \Pr\{T = 1/E_{i1} = 0, E_{i2} = 0, \ldots, E_{in_i} = 0\}.$ (4.35)

4.6 Computer Program and Computational Results

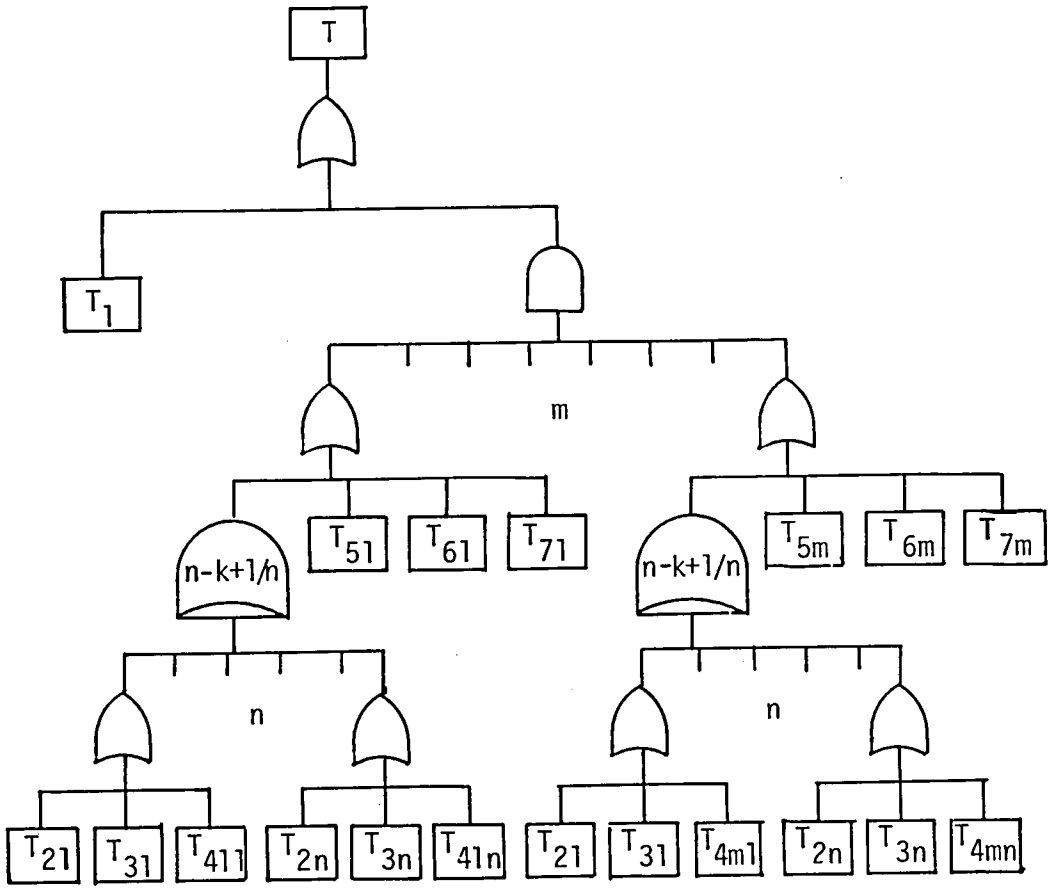
Computer program ALFAT (ALgorithm for evaluating FAult Tree) has been constructed in FORTRAN for implementing the method presented in this chapter. ALFAT is composed of three parts, FTREP, SEQTEP, PROBTE. FTREP yields reverse Polish sequence P(1001) and tree sequence TS(1001) of top event 1001 by the method of Section 2.3. SEQTEP yields reverse Polish sequence PS(1001) and tree sequence TSS(1001) of the symbolic form of top event probability from P(1001) and TS(1001) by the method of Section 4.3. PROBTE computes the top event probability and sensitivity coefficients of primary events from PS(1001) and TSS(1001) by the method of Section 4.4. The list of top event or each intermediate event must be given as data input along with the number of sons, members of sons (listed in order from the leftmost son) and the logic gate. The probabilities of occurrence of primary events must also be given. The computer output for the example of Fig. 14 is shown in Fig. 17.

The computational results are demonstrated for the containment spray injection system located in PWR nuclear power plant [68]. This system functions so as to reduce the pressure in containment when a large LOCA (Loss of Coolant Accident) occurred. This system is composed of trip circuit (TC) and containment spray injection subsystem (CSIS). The TC detects high pressure in containment and makes a signal to start the valves and pumps of the CSIS. The CSIS delivers cold water containing boron from the refueling water storage tank (RWST) to the containment volume.

Consider the case where n TCs are used in k/n redundancy and m CSISs are used in parallel to improve system reliability. Fig. 18 shows the fault tree structure for top event T "insufficient fluid flows in containment". Table 6 shows system unavailabilities (top event probabilities) computed for several values of (m, k/n), given unavailabilities of primary events [68].

```
EXISTENCE OF MUTUALLY EXCLUSIVE EVENTS =    0

LIST OF TOP EVENT AND INTERMEDIATE EVENTS

 1001 = 10021003  -3
 1002 = 10041005  -2
 1003 =     6    31006  -2
 1004 =    11007  -3
 1005 =     2     4     5  -3
 1006 =     4     7     8     9  -5
 1007 =     2     3  -2

NUMBER OF PRIMARY EVENTS =       9

REVERSE POLISH SEQUENCE OF TOP EVENT

  1  2  3 -2 -3  2  4  5 -3 -2 *   6  3  4  7  8  9 -5 -2 -3

TREE SEQUENCE OF TOP EVENT

  1  1  1 -1 -1  1  1  1 -2 -1 *   1  1  1  1  1  1 -3 -2 -1

LENGTH OF SEQUENCES OF TOP EVENT =      19

REVERSE POLISH SEQUENCE OF TOP EVENT PROBABILITY

    1     2     4     5    -3    -2  1000     3    -1    -2
    1     5    -2  1000     2    -1    -2     2     0     6
    7     8     9    -4    -2    -3  1000     4    -1    -2
    1     2    -3     6     7     8     9    -3    -2    -3
    4    -2     0     3    -2     0

TREE SEQUENCE OF TOP EVENT PROBABILITY

    1     1     1     1    -2    -1     1     1    -1    -1
    1     1    -1     1     1    -1    -1     1    -1     1
    1     1     1    -2    -1    -1     1     1    -1    -1
    1     1    -1     1     1     1     1    -2    -1    -1
    1    -1    -1     1    -1    -1

LENGTH OF SEQUENCES OF TOP EVENT PROBABILITY =      46
```

Fig. 17 Computer Output for the Example of Fig. 14



T : insufficient fluid flows in containment (top event)

T_1 : rupture of RWST

$T_{21} \sim T_{2n}$: malfunction of TC

$T_{31} \sim T_{3n}$: failure of power supply of TC

$T_{411} \sim T_{4mn}$: malfunction of input relay

$T_{51} \sim T_{5m}$: failure of CSIS (containing malfunction of output relay)

$T_{61} \sim T_{6m}$: failure of power supply of CSIS and output relay

$T_{71} \sim T_{7m}$: system down of CSIS due to test and maintenance

Fig. 18 Fault Tree Structure of Containment Spray Injection System of PWR

Table 6 System Unavailabilities

k/n \ m	1	2	3
1/1	2.5×10^{-2}	1.9×10^{-3}	1.3×10^{-3}
1/2	2.3×10^{-2}	5.3×10^{-4}	1.7×10^{-5}
2/3	2.3×10^{-2}	5.8×10^{-4}	5.9×10^{-5}
3/4	2.3×10^{-2}	6.2×10^{-4}	1.0×10^{-4}

Table 7 Importances and Sensitivity Coefficients of Primary Events

primary event	unavailability	(1, 1/1)	(2, 1/1)	(2, 3/4)	(2, 3/4)
		importance	importance	importance	sensitivity coefficient
T_1	4.4×10^{-7}	1.7×10^{-5}	2.4×10^{-4}	7.0×10^{-4}	1.0
$T_{21} \sim T_{2n}$	1.3×10^{-3}	4.9×10^{-2}	6.7×10^{-1}	8.5×10^{-3}	4.1×10^{-3}
$T_{31} \sim T_{3n}$	4.1×10^{-5}	1.6×10^{-3}	2.2×10^{-2}	6.6×10^{-2}	1.0
$T_{411} \sim T_{4mn}$	1.4×10^{-3}	5.5×10^{-2}	1.7×10^{-2}	4.4×10^{-4}	2.0×10^{-4}
$T_{51} \sim T_{5m}$	1.9×10^{-2}	5.3×10^{-1}	1.7×10^{-1}	7.0×10^{-1}	2.3×10^{-2}
$T_{71} \sim T_{7m}$	4.1×10^{-3}	1.6×10^{-1}	5.1×10^{-2}	1.5×10^{-1}	2.3×10^{-2}

(Importances of $T_{61} \sim T_{6m}$ are same as those of $T_{31} \sim T_{3n}$.)

Table 7 shows criticality importances of primary events computed for three values of (m, k/n) and sensitivity coefficients of primary events for (2, 3/4). From these tables, we can know the following facts.

1. Little improvement of system unavailability is made even if increasing the redundancy of TC in keeping only one CSIS.

2. Appreciable improvement of system unavailability are made by using two CSISs in keeping one TC, but further increase of CSIS makes little improvement.

3. Appreciable improvement of system unavailability is made by increasing the redundancy of TC in keeping two CSISs. There is little difference of the degree of improvement among 1/2, 2/3. 3/4 redundancies. However, it would be effective to use 2/3 or 3/4 redundancy from the viewpoint of reducing the possibility of "spurious trip" which means the situation that the system functions in the normal state.

4. The improvement process of (1, 1/1) → (2, 1/1) → (2, 3/4) is coincided with the process of adding redundant units to the greater of criticality importances of TC and CSIS. This fact implies that the criticality importance is an useful measure for reliability improvement.

Table 8 shows the comparison of computation times between the case of using k/n gates and the case of transforming each k/n gate into an equivalent tree containing only AND & OR gates. This table indicates that the use of k/n gates appreciably reduces the computation time.

For several large fault trees, ALFAT was tested on a FACOM 230-75. The computation time of top event probability was 2.2 sec for an example (119, 30, 15), 7.9 sec for an example (299, 100, 15) and 100.2 sec for an example (299, 100, 25), where in each parenthesis, the first number is the length of sequences of top event, the second number the total number of primary events and the third the total number of repeated primary events.

Table 8 Comparison of Computation Time

k/n	m	number of primary events	case A		case B	
			length of sequences	computation time (sec)	length of sequences	computation time (sec)
2/3	1	11	19	0.19	34	0.99
2/3	2	16	37	0.88	67	2.09
2/3	3	21	54	1.56	99	3.39
3/4	1	13	23	0.29	61	3.34
3/4	2	19	45	1.45	121	5.22
3/4	3	25	66	2.27	180	8.44

case A : case of using k/n logic gates

case B : case of using only AND gates and OR gates

(tested on FACOM 230-38)

Chapter 5

A Method for Obtaining Sequence Representation of Transmission Boolean Function of Network

In this chapter a method is presented for representing the transmission Boolean function of a network by the reverse Polish sequence and tree sequence. Such a representation is applicable to various reliability evaluations and has an advantage of saving memories and computation time for executing reliability calculation by a computer. Computational results are given for several examples.

5.1 Introduction

Many methods have been presented for calculating the node-pair reliability of network. Most of existing methods are based on the strategy of calculating only node-pair reliability directly through state enumeration, path (cutset) enumeration, reduction, decomposition etc. The state enumeration [60], path enumeration [10, 16, 67] and cutset enumeration methods [30] are most popular, but requires a huge amount of computation for a large network. The typical reduction methods are Misra's algorithm [37] treating only the reduction of series-parallel subnetwork and Krishnamurthy & Komissar's

algorithm [33] applicable to non-series-parallel network. A typical decomposition method is Hänsler's algorithm [24] applicable to only undirected network.

Instead of those methods, the method presented here is based on the strategy of first obtaining the transmission Boolean function and then executing reliability evaluations.

It is of significance to store the transmission Boolean functions for node-pairs, because they can not only be used for the calculations of node-pair reliabilities, but also be applied to various evaluations, such as a) calculations of reliability measures concerning several specific nodes (e.g., the probability that we can communicate from a central node to some other nodes) and b) reliability evaluations for the case where nodes and branches are mutually dependent (see Example 2 of Section 1.3).

A sum-of-product form of transmission Boolean function can be obtained by using minimal paths [10], but such a form is often too long to execute reliability evaluation efficiently. Our object is to represent the transmission Boolean function as shortly as possible by the reverse Polish sequence and tree sequence. Shortening these sequences is useful for saving memories and computation time for reliability calculation.

The principle of method is to repeat recursively two processes, i.e., reduction of network and decomposition of completely reduced network, and to reduce to obtaining sequence representations of series-parallel subnetworks. A backtracking technique is used for implementing this method efficiently.

Section 5.2 explains basic processes of method. Section 5.3 presents the overall method and shows the illustration of method by an example. Section 5.4 presents the computational results for several examples.

5.2 Basic Processes of Method

5.2.1 Preliminaries

Consider a network consisting of n nodes numbered by 1, 2, ..., n and m directed branches numbered by $n+1$, $n+2$, ..., $n+m$, where the source node is 1 and the sink node is n.

A *node variable* or *branch variable* v_i is defined as a binary variable attached to node i or branch i. The value of v_i is 1 if i is in the operating state and 0 if i is in the failed state.

Transmission variable T is 1 if there exists at least one path from the source node to the sink node in which all nodes and branches are in the operating state, and 0 otherwise.

Then, transmission variable T can be represented by a Boolean function of node variables and branch variables, which is called the *transmission Boolean function* (TBF) of network [10].

The TBF is represented by reverse Polish sequence P(T) and tree sequence TS(T) to make the computer processing easier. These sequences are fundamentally same as reverse Polish sequence and tree sequence of top event presented in Chapter 2 which represent the Boolean function relating the top event to primary events.

The following subsections explains three basic processes of the method for obtaining P(T) and TS(T), i.e., sequences of series-parallel structure, decomposition of network and reduction of network.

5.2.2 Sequences of Series-Parallel Structure

The TBF of series network is expressed as the logical product of node variables and branch variables. The reverse Polish sequence of TBF is obtained by putting node and branch numbers in order of appearing in the path from the source node to the sink node and lastly putting AND gate. The tree sequence of TBF is obtained by changing each node or branch number of reverse Polish sequence into

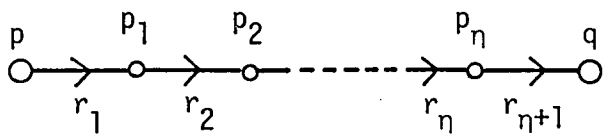
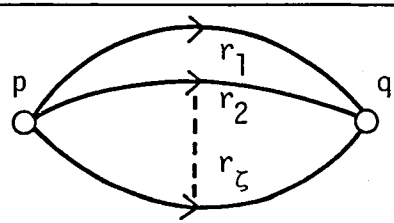
	Series Structure	Parallel Structure
Network		
Transmission Boolean Function	$T = r_1 \wedge p_1 \wedge r_2 \wedge p_2 \wedge \cdots \wedge r_\eta \wedge p_\eta \wedge r_{\eta+1}$	$T = r_1 \vee r_2 \vee \cdots \vee r_\zeta$
P(T) TS(T)	$P(T) = r_1 p_1 r_2 p_2 \cdots r_\eta p_\eta r_{\eta+1} \wedge$ TS(T) = 1 1 1 1 ....1 1 1 -2η	$P(T) = r_1 r_2 \cdots r_\zeta \vee$ TS(T) = 1 1 .....1 -(ζ-1)

Fig. 19 Sequence Representation of Series or Parallel Structure

value +1 and AND gate into value -2η, where η is the number of nodes excluding the source node and the sink node.

The TBF of parallel network is expressed as the logical sum of branch variables. The reverse Polish sequence of TBF is obtained by putting branch numbers and lastly putting OR gate. The tree sequence of TBF is obtained by changing each branch number of reverse Polish sequence into value +1 and OR gate into value $-(\zeta - 1)$, where ζ is the number of branches.

Fig. 19 shows the TBF and P(T) & TS(T) for series or parallel structure. In this chapter, symbols $\wedge$, $\vee$ are used for representing AND & OR gates in reverse Polish sequence. But, in the computer processing, they are represented by -2, -3 similar to Chapter 2.

The TBF of a series-parallel network is expressed as a combination of logical product terms and logical sum terms. The TBF of the network of Fig. 20a is

$$T = v_5 v_2 v_6 \vee (v_7 \vee v_8) v_3 v_9. \tag{5.1}$$

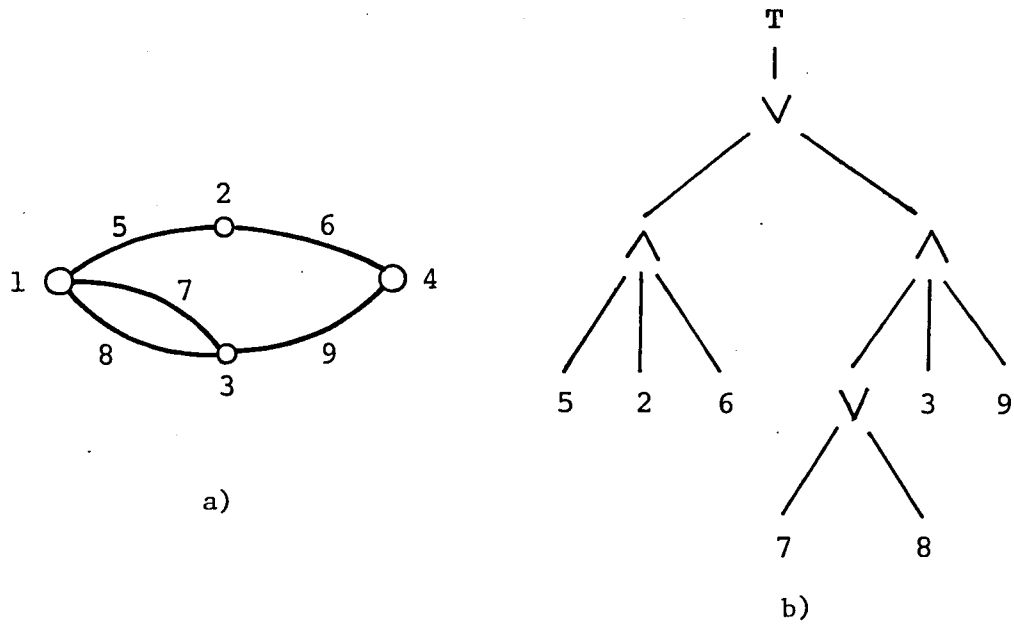


Fig. 20 Example of Series-Parallel Network and Tree Structure Equivalent to TBF

This TBF is equivalent to a tree having node & branch numbers as leaves and logic operators as branch nodes such as shown in Fig. 20b. Therefore, the sequences P(T) & TS(T) of this TBF is obtained by repeating the substitution of the sequences of series structure or parallel structure as follows.

$$\begin{aligned} P(T) &= 5\ 2\ 6\ \wedge\ 7\ 8\ \vee\ 3\ 9\ \wedge\ \vee \\ TS(T) &= 1\ 1\ 1\ -2\ 1\ 1\ -1\ 1\ 1\ -2\ -1 \end{aligned} \tag{5.2}$$

5.2.3 Decomposition of Network

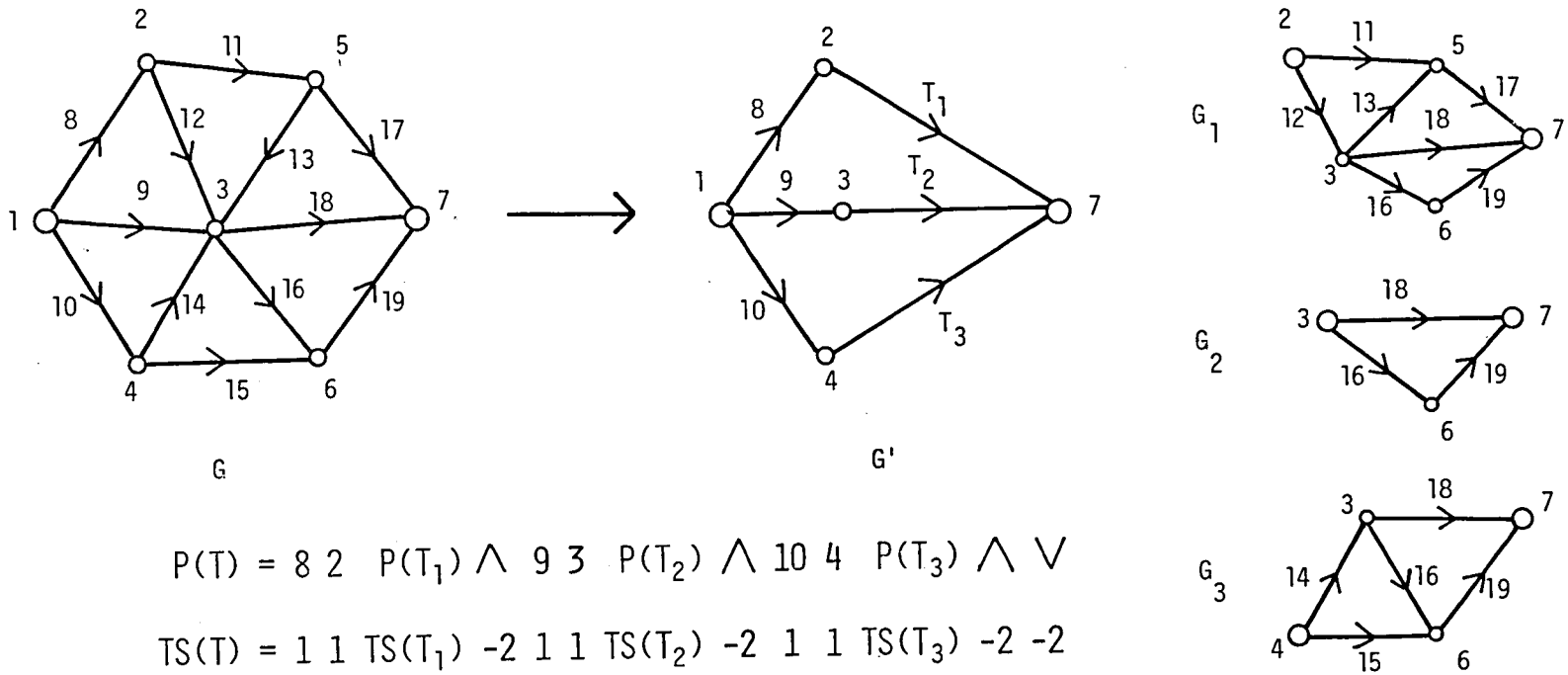
Let i and j be the source node and sink node of a network G, respectively, and suppose that there exist k branches $b_1 = (i, i_1)$, $b_2 = (i, i_2)$, ..., $b_k = (i, i_k)$ incident out of i. Let T be the transmission variable of G and T_γ, $\gamma = 1, 2, ..., k$ be the transmission variables of the subnetworks G_γ with source node i_γ and sink node j obtained by removing, from G, node i, all branches that initiate or terminate at node i and other irrelevant branches & nodes which means branches & nodes that do not appear in any minimal path from i_γ to j. Then

$$T = v_{b_1} v_{i_1} T_1 \vee v_{b_2} v_{i_2} T_2 \vee ..... \vee v_{b_k} v_{i_k} T_k. \tag{5.3}$$

P(T) and TS(T) are

$$\begin{aligned} P(T) &= b_1\ i_1\ P(T_1)\ \wedge\ b_2\ i_2\ P(T_2)\ \wedge\ ....\ b_k\ i_k\ P(T_k)\ \wedge\ \vee \\ TS(T) &= 1\ 1\ TS(T_1)\ -2\ 1\ 1\ TS(T_2)\ -2\ ....\ 1\ 1\ TS(T_k)\ -2\ -(k-1) \end{aligned} \tag{5.4}$$

Thus the problem of obtaining the sequences of TBF of G is decomposed into the problems of obtaining the sequences of TBFs of subnetworks G_1, G_2, ..., G_k.



$$P(T) = 8\ 2\ \ P(T_1) \wedge\ 9\ 3\ \ P(T_2) \wedge\ 10\ 4\ \ P(T_3) \wedge \vee$$

$$TS(T) = 1\ 1\ TS(T_1)\ -2\ 1\ 1\ TS(T_2)\ -2\ \ 1\ \ 1\ TS(T_3)\ -2\ -2$$

Fig. 21 Example of Decomposition

Fig. 21 illustrates this decomposition process by an example of network. In the network G, there exist three branches 8, 9, 10 incident out of source node 1. Therefore, G can be transformed into an equivalent series-parallel network G'. Branch variables T_1, T_2, T_3 are equivalent to transmission variables of subnetworks G_1, G_2, G_3, respectively. Thus the problem of obtaining the TBF of G is decomposed into the problems of obtaining the TBFs of G_1, G_2, G_3.

5.2.4 Reduction of Network

The reduction is the process of finding two-terminal subnetworks and replacing them with equivalent branches. There are two types of reduction; Type I is the case of finding series-parallel two-terminal subnetworks and type II is the case of finding non-series-parallel two-terminal subnetworks. Each type can also be classified into a) single-directional case and b) bi-directional case. Fig. 22 illustrates the examples of these types of reduction.

By the reduction, the problem of obtaining the TBF of network is partitioned into the problems of obtaining the TBFs of the reduced network and two-terminal subnetworks found.

If a network contains no two-terminal subnetwork, then it is called a *completely reduced network.*

In what follows, the algorithms for implementing the reduction process are described.

The reduction is implemented after eliminating all irrelevant nodes and branches from the network. First type I is implemented and then type II is implemented.

Algorithm for the Reduction of Type Ia:

The reduction of type Ia is implemented by applying Misra's algorithm [37].

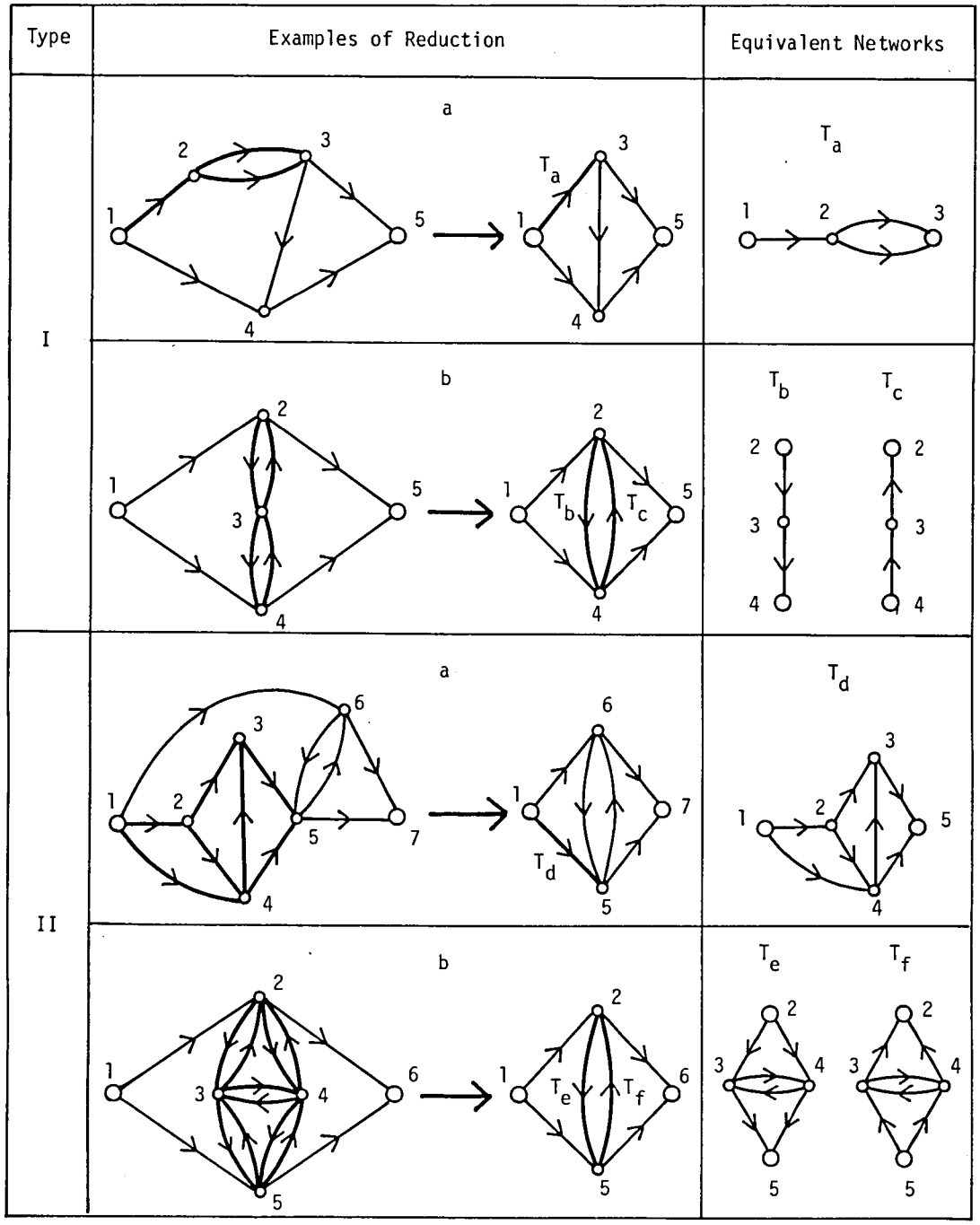


Fig. 22 Types of Reduction

Check each node if it has indegree = outdegree = 1, where the indegree (outdegree) of a node is the number of branches incident into (out of) that node.

If such node i is found, then there exists the series subnetwork composed of branch $b_1 = (i_a, i)$, node i and branch $b_2 = (i, i_b)$. Then replace this series subnetwork with a new branch $b_3 = (i_a, i_b)$ whose variable is equivalent to $v_{b_1} \wedge v_i \wedge v_{b_2}$.

If branch b_3 is in parallel with another branch $b_4 = (i_a, i_b)$, then replace these branches i_3, i_4 with a new branch $i_5 = (i_a, i_b)$ whose variable is equivalent to $v_{b_3} \vee v_{b_4}$.

Repeat the above process for the resulting network until the node that has indegree = outdegree = 1 has been exhausted.

Algorithm for the Reduction of Type Ib:

The reduction of type Ib is implemented by modifying the algorithm for type Ia.

Check each node if it has indegree = outdegree = 2. If such node i is found and only two nodes are adjacent with node i through branches $b_1 = (i_a, i)$, $b_2 = (i, i_a)$, $b_3 = (i_b, i)$, $b_4 = (i, i_b)$, then eliminate the subnetwork composed of node i, branches b_1, b_2, b_3, b_4 and add new branches $b_5 = (i_a, i_b)$, $b_6 = (i_b, i_a)$ whose variables are equivalent to $v_{b_1} \wedge v_i \wedge v_{b_4}$, $v_{b_2} \wedge v_i \wedge v_{b_3}$, respectively.

If branch b_5 (or b_6) is in parallel with another branch $b_7 = (i_a, i_b)$ (or $b_8 = (i_b, i_a)$), then replace branches b_5, b_7 (or b_6, b_8) with a new branch $b_9 = (i_a, i_b)$ (or $b_{10} = (i_b, i_a)$) whose variable is equivalent to $v_{b_5} \vee v_{b_7}$ (or $v_{b_6} \vee v_{b_8}$).

Repeat the above process for the resulting network until the node that has indegree = outdegree = 2 has been exhausted.

It is not so much time-consuming to find only series-parallel two-terminal subnetworks, because the above algorithms contain only the checks for nodes. But it is time-consuming to find all two-terminal subnetworks containing non-series-parallel case, because we must check the existence of two-terminal subnetwork for all node-pairs. In this case, only one existing method is Krishnamurthy and Komissar's algorithm [33], but their algorithm cannot find all two-terminal subnetworks since it uses only reachability matrix. So presented is a new algorithm which can find all two-terminal subnetworks .

The algorithm uses the property that any node (excluding two terminals) of a two-terminal subnetwork is adjacent to only nodes of the subnetwork itself.

Algorithm for Finding all Two-Terminal Subnetworks Containing Type II:

Check, for each node-pair (i, j), the existence of two-terminal subnetwork having nodes i, j as two terminals by the following procedure.

Let $b_{i\gamma}$, $\gamma = 1, 2, \ldots, m_i$ be the branches incident out of node i and $b_{j\kappa}$, $\kappa = 1, 2, \ldots, m_j$ be the branches incident out of node j. Obtain, for each $b_{i\gamma}$(or $b_{j\kappa}$), set $S_{i\gamma}$ (or $S_{j\kappa}$) of nodes contained in minimal paths from i to j (or from j to i) through $b_{i\gamma}$ (or $b_{i\kappa}$). Let $S_i \equiv \{S_{i1}, S_{i2}, \ldots, S_{im_i}\}$, $S_j \equiv \{S_{j1}, S_{j2}, \ldots, S_{jm_j}\}$ and $S \equiv \{S_{i1}, \ldots, S_{im_i}, S_{j1}, \ldots, S_{jm_j}\}$.

Take up a subset Q of S and let P be the union of elements of subset Q. Then check if any node of P is adjacent to only nodes of $P \cup \{i, j\}$. If this is false, then repeat the checks for remaining subsets Q of S. If this is true, then the subnetwork composed of nodes of $P \cup \{i, j\}$ is a two-terminal subnetwork. Then replace this subnetwork with new branches $b_i = (i, j)$, $b_j = (j, i)$ if Q is composed of elements of both S_i and S_j. If Q is composed

of elements of only S_i (or S_j), then replace the subnetwork with a new branch $b_i = (i, j)$ (or $b_j = (j, i)$).

Continue the above process for the resulting network until two-terminal network has been exhausted for any node-pair.

5.3 Overall Method

The overall method repeats recursively the decomposition of completely reduced network by the method of 5.2.3 and the reduction of reducible network by the method of 5.2.4 until reaching series-parallel structures. The method is implemented by a backtracking technique.

The overall method is illustrated by an example.

Fig. 23 is the illustration of the procedure of method for the network with 7 nodes and 12 branches. A backtracking technique implements basic processes in the order put in Fig 23. Each new branch is represented by the number more than 100.

Sequences P(T), TS(T) are obtained by substituting the sequence obtained in each process into the corresponding place. They are shown at the uppermost of Fig. 24.

For comparison, Fig. 24 also shows the sequences obtained by using only decomposition without using reduction and the sequences obtained by using minimal paths.

From Fig. 24, it is apparent that the length of sequences can appreciably be shortened by using both decomposition and reduction.

5.4 Computational Results

A computer program (program A) has been developed in FORTRAN for implementing the method presented in this chapter. To test the efficiency of reduction process, the method of obtaining the sequences of TBF by repeating only decomposition without using

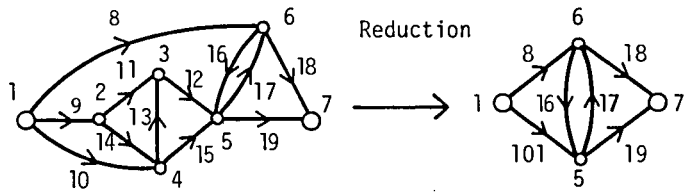
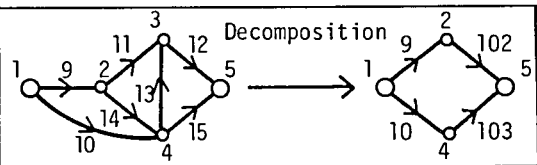
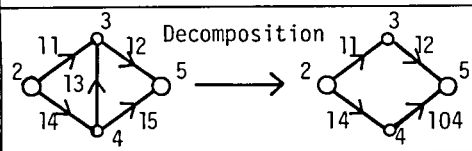
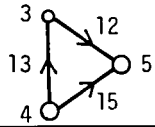
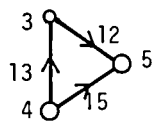
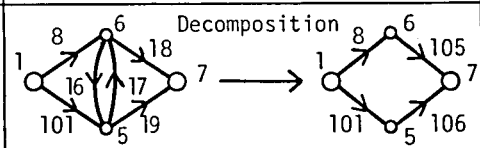
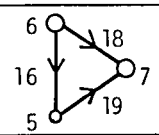
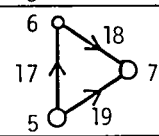
	Which Stage ?	Reverse Polish Sequence Tree Sequence
T		
101		P(101) = 9 2 P(102) Λ 10 4 P(103) Λ V TS(101) = 1 1 TS(102) -2 1 1 TS(103) -2 -1
102		P(102) = 11 3 12 Λ 14 4 P(104) Λ V TS(102) = 1 1 1 -2 1 1 TS(104) -2 -1
104		P(104) = 13 3 12 Λ 15 V TS(104) = 1 1 1 -2 1 -1
103		P(103) = 13 3 12 Λ 15 V TS(103) = 1 1 1 -2 1 -1
T		P(T) = P(101) 5 P(106) Λ 8 6 P(105) Λ V 7 1 Λ TS(T) = TS(101) 1 TS(106) -2 1 1 TS(105) -2 -1 1 1 -2
105		P(105) = 16 5 19 Λ 18 V TS(105) = 1 1 1 -2 1 -1
106		P(106) = 17 6 18 Λ 19 V TS(106) = 1 1 1 -2 1 -1

Fig. 23 Illustration of Method by an Example

Using Both Decomposition and Reduction Length of Sequences = 48

```
P(T) = 9 2 11 3 12 ∧ 14 4 13 3 12 ∧ 15 ∨ ∧ ∨ ∧ 10 4 13 3 12 ∧ 15 ∨ ∧ ∨ 5 17 6 18 ∧ 19 ∨ ∧
TS(T) = 1 1 1 1 1 -2 1 1 1 1 1 -2 1 -1 -2 -1 -2 1 1 1 1 1 -2 1 -1 -2 -1 1 1 1 1 -2 1 -2 -3

        8 6 16 5 19 ∧ 18 ∨ ∧ ∨ 7 1 ∧
        1 1 1 1 1 -2 1 -1 -2 -1 1 1 -2
```

Using Only Decomposition Length of Sequences = 76

```
P(T) = 8 6 16 5 19 ∧ 18 ∨ ∧ 9 2 11 3 12 5 17 6 18 ∧ 19 ∨ ∧ 14 4 13 3 12 5 17 6 18 ∧ 19 ∨ ∧
TS(T) = 1 1 1 1 1 -2 1 -1 -2 1 1 1 1 1 1 1 1 -2 1 -1 -4 1 1 1 1 1 1 1 1 1 -2 1 -1 -4

        15 5 17 6 18 ∧ 19 ∨ ∧ ∨ ∧ ∨ ∧ 10 4 13 3 12 5 17 6 18 ∧ 19 ∨ ∧ 15 5 17 6 18 ∧ 19 ∨ ∧
        1 1 1 1 1 -2 1 -1 -2 -1 -2 -1 -2 1 1 1 1 1 1 1 1 1 -2 1 -1 -4 1 1 1 1 1 -2 1 -1 -2

        ∨ ∧ ∨ 7 1 ∧
        -1 -2 -2 1 1 -2
```

Using Minimal Paths Length of Sequences = 104

```
P(T) = 8 6 18 ∧ 8 6 16 5 19 ∧ 9 2 11 3 12 5 17 6 18 ∧ 9 2 11 3 12 5 19 ∧ 9 2 14 4 13 3 12
TS(T) = 1 1 1 -2 1 1 1 1 1 -4 1 1 1 1 1 1 1 1 1 -8 1 1 1 1 1 1 1 -6 1 1 1 1 1 1 1

        5 17 6 18 ∧ 9 2 14 4 13 3 12 5 19 ∧ 9 2 14 4 15 5 17 6 18 ∧ 9 2 14 4 15 5 19 ∧ 10 4
        1 1 1 1 -10 1 1 1 1 1 1 1 1 1 -8 1 1 1 1 1 1 1 1 1 -8 1 1 1 1 1 1 1 -6 1 1

        13 3 12 5 17 6 18 ∧ 10 4 13 3 12 5 19 ∧ 10 4 15 5 17 6 18 ∧ 10 4 15 5 19 ∧ ∨ 7 1 ∧
        1 1 1 1 1 1 1 -8 1 1 1 1 1 1 1 -6 1 1 1 1 1 1 1 -6 1 1 1 1 1 -4 -11 1 1 -2
```

Fig. 24 Comparison of Sequences of TBF for Example of Fig. 23

reduction has also been programmed (program B). The computational results for several examples are shown in Table 9. The node-pair reliability for each example was calculated from P(T) and TS(T) by using the method presented in Chapter 4.

Rouphly speaking from my experience, the node-pair reliability can be calculated in several minutes on FACOM M-190 computer for the networks having nodes and branches less than 100.

As for the length of sequences, program A yields much shorter sequences than program B. As for the computation time for obtaining only sequences of TBF, program A requires longer computation time than program B. But, as for the computation time containing reliability calculation, program A requires shorter computation time than program B.

The efficiency of reduction varies with the topologies of networks. As example 3 shows, the computation time for program A is, in the best case, several 10's order shorter than program B.

Table 9 Computational Results

Example	Number of nodes	Number of branches	Using both decomposition and reduction			Using decomposition only		
			L	t_s	t_r	L	t_s	t_r
1	10	36	81	0.2	0.3	204	0.2	0.6
2	15	31	154	1.2	4.7	583	0.3	13.6
3	20	35	119	2.0	2.8	1191	0.5	163

L: length of P(T) and TS(T) t_s (sec): computation time required to obtain P(T) and TS(T) t_r (sec): total computation time containing reliability calculation (tested on FACOM M-190 computer)

Chapter 6

Optimal Design of a System with Time-Dependent Reliability

This chapter formulates an optimal design problem for a series system with time-dependent reliability. The variables for optimization are the number of redundant components in each subsystem and the mean-time-to-failure of each component. There is a cost-constraint. The time for which the system reliability exceeds a specified value is to be maximized. Similarly the cost could be minimized for a constraint on the mission time and reliability. A solution method for the formulated problems is presented along with an example.

6.1 Introduction

A system designer is often asked to design a high reliable system. He usually considers two ways of improving system reliability: 1) adding redundant components and 2) increasing reliability of a component. The latter way corresponds to tightening quality control in producing each component, developing new components with higher reliability, etc. Both ways usually bring the increase of system cost with them. Therefore, at the stage of designing a high

reliable system, one of important problems that he is faced with is how to make tradeoffs between reliablity and cost.

The problems of optimizing system reliablity under given constraints are classified from the following viewpoints.

A. Which is the problem, 1) single-objective or 2) multi-objective ?

B. Which is the system reliability function, 1) separable or 2) non-separable in subsystem reliabilities ?

C. Which are considered as the variables for optimization, 1)only one or 2) both of two ways of improving reliability ?

Since the publication of Moskowitz and McLean's paper [40], the studies on system reliability optimization problems have mainly been focused on A1-B1-C1 problems. The papers [43] & [44] are the latest works on approximate and exact methods, respectively, for solving chiefly the optimal redundancy allocation problem which is a pure-integer programming problem. In Section 6.2, these works are summarized. Lately some researchers are beginning to be interested in A1-B1-C2 [38, 50, 66], A1-B2-C1 [1, 11], A2 [46, 63, 29] problems, too. This chapter is given to formulate and solve one of A1-B1-C2 problems.

Only a few researchers [38, 66] consider the problem of determining both optimal number of redundant components and optimal component reliability which is a mixed-integer programming problem, but they assume time-independent reliability. This chapter formulates the problem of optimizing both ways of improving system reliability under time-dependent reliability. System reliability is monotonically decreasing with mission time. For this time-dependency, we adopt as the performance index the mission time that the system reliability is above a preassigned value. The solution methods have been presented for the problems of maximizing the above mission time under system cost constraint and minimizing the system cost under mission time constraint.

6.2 Optimal Redundancy Allocation Problem and Its Solution Methods

6.2.1 Optimal Redundancy Allocation Problem

The optimal redundancy allocation problem of a series system with n subsystems is formulated as the problem of finding the vector $X = (x_1, x_2, \ldots, x_n)$ so as to maximize the system reliability

$$R_S = \Pi_{i=1}^{n} R_i(x_i) = \Pi_{i=1}^{n} \{1 - (1 - p_i)^{x_i}\}$$

subject to r constraints

$$\Sigma_{i=1}^{n} g_{ji}(x_i) \leq b_j \qquad \text{for } j = 1, 2, \ldots, r,$$

where p_i and x_i are the reliability of a component and the number of parallel components, respectively, in subsystem i, and $g_{ji}(x_i)$ is the amount of resource j (cost, volume, weight, etc.) consumed at subsystem i. The constraints are separable and are monotonically increasing with x_i.

This is a pure-integer programming problem. Many methods have been presented for solving this problem [45, 65]. They are classified into two groups: approximate methods that provide an approximate, often exact, solution but can be solved within a reasonable amount of computation time and exact methods that require a considerable amount of computation time. In this section, an approximate (heuristic) method [43] and an exact method [44] are summarized; these two methods represent the current state of the techniques for solving the optimal redundancy allocation problem.

6.2.2 Approximate Solution Method

The objects of devising an approximate method are:

1) to obtain a near-optimal, often optimal, solution for a large problem, since an exact method cannot solve it in a reasonable

amount of computation time, and/or

2) to use the approximate solution as an initial solution for an exact method.

In the present method, the solution is obtained by repeatedly adding one redundant component to the subsystem that has the greatest value of the 'weighted sensitivity function' without violating any of the constraints. The weighted sensitivity function for each subsystem is the product of a quantity obtained as a function of the objective function and the constraints. The balance between these two quantities is controlled by a balancing coefficient α.

Let x_i^c be the current solution of x_i. Then, the weighted sensitivity function s_i of subsystem i is defined equationally as follows.

$$s_i \equiv \Delta f_i(x_i^c + 1) \cdot [(1 - \alpha) \cdot \min_{k \in L_{+1}} \{\Delta x_k\} + \alpha \cdot \Delta x_i]$$

where

$$\Delta f_i(k) \equiv \ln R_i(k) - \ln R_i(k-1) \qquad (R_i(0) \equiv 1)$$

$$\Delta g_{ji}(k) \equiv g_{ji}(k) - g_{ji}(k-1) \qquad (g_{ji}(0) \equiv 0)$$

$$b_j^c \equiv b_j - \sum_{i=1}^{n} \sum_{k=1}^{x_i^c} \Delta g_{ji}(k)$$

$$\Delta x_i \equiv \min_j \{b_j^c / \Delta g_{ji}(x_i^c + 1)\}$$

$$L_{+1} \equiv \{i \mid \Delta x_i \geq 1\}.$$

One redundant component is added to subsystem k such that

$$s_k = \max_{i \in L_{+1}} \{s_i\},$$

i.e., $x_k^c \leftarrow x_k^c + 1$ is reset, and the similar operation is repeated.

The procedure begins with the initial current solution X^c = (1, 1, ..., 1) and ends when L_{+1} becomes empty. The solutions for a set of α (probably α = 0, 0.1, ..., 0.9, 1.0; $1/\alpha$ = 0.9, 0.6, 0.3) should be obtained. The best solution among the solutions for the given set of α's is the final solution.

The computational results for the present method and the other methods have been compared in [43]. The computation time was 0.7 sec and 3.1 sec on a FACOM 230/75 digital computer for linear-constraint problems with n = 15, r = 2 and n = 30, r = 3, respectively. The present method can also be applied for the case where a more reliable component is used to improve the reliability of some subsystem, instead of redundant components being added.

6.2.3 Exact Solution Method

The solution methods that guarantee exact optimality use either of dynamic programming and integer programming. The method using dynamic programming becomes impractical for problems having more than three functional constraints. In the method using integer programming, an increase in the number of constraints affects very little the size of problem, but the computation time increases exponentially with respect to the number of variables. In general the latter is superior to the former when the problem has multi-constraints.

Most of methods using integer programming applies the existing algorithm after reformulating the problem into a 0-1 programming problem. Those methods can not solve a large problem (limited to somewhat less than 10 variables), mainly because the numbers of constraints and variables increase when 0-1 algorithms are used. The present method is based on branch-and-bound and is designed for dealing with integer variables as they are, without increasing the number of variables or constraints.

Branch-and-bound is an optimization technique that uses the

tree enumeration. Each branch from a node to its immediate successors corresponds to a restriction of fixing a variable v_k to one allowable integer value α_k. Each node corresponds to a problem; the top node v_0 corresponds to the original problem, and generally the subproblem corresponding to node v_κ ($\kappa = 1, 2, 3, \ldots$) is

Subproblem (P_κ): maximize R_S subject to

$$\Sigma_{i=1}^{n} g_{ji}(x_i) \leq b_j \qquad (j = 1, 2, \ldots, r),$$

$$x_i = \alpha_i \quad (i \in L^r), \quad \underline{x}_i \leq x_i \leq \bar{x}_i; \; x_i \text{ integer } (i \in L^f),$$

where L^r is the set of indices of fixed variables restricted by the branches along the unique path from v_0 to v_κ, and L^f is the set of indices of the other variables, i.e., free variables.

Visits to nodes are made in the preorder. If there are no nodes that need further branching, the enumeration is complete.

The manners of selecting a fixed variable x_k from among free variables at node v_κ and deciding both bounds $\underline{x}_k$, $\bar{x}_k$ of x_k greatly influence the efficiency of enumeration. We fix variables beforehand in the order of increasing difference between both bounds for variables obtained from the constraints at v_κ. Both bounds of x_k are obtained from the solution of a relaxation problem for P_κ. The relaxation problem varies with the character specific to a given problem. If the solution of the relaxation problem for P_κ is worse than the current incumbent, then no further branching from v_κ is made. The solution of the approximate method presented in 6.2.2 is used as the initial incumbent, and the incumbent is changed whenever a better solution is obtained.

The computation time was 4.9 sec and 109 sec on a FACOM 270/75 computer for linear-constraint problems with $n = 15$, $r = 2$ and $n = 30$, $r = 3$, respectively (same as those in 6.2.2). The method can be applied to non-separable problems by appropriately selecting the relaxation problem.

6.3 Preliminaries

The notation used after this section is listed here. Some other notation is used and defined locally.

n	number of series subsystem.
i	subsystem number, $i = 1, 2, \ldots, n$.
S_i	subsystem i.
$R_S(t)$	system reliability at time t.
$R_i(t)$	reliability of S_i at time t.
$p_i(t)$	reliability of a component in S_i at time t.
C_S	system cost.
τ_i	mean time to failure of a component in S_i.
x_i	number of parallel components in S_i; the number of redundant components is $x_i - 1$.
τ_{i0}	initial value of τ_i in the optimization procedure.
$\boldsymbol{\tau}$, $\boldsymbol{X}$	vectors $(\tau_1, \tau_2, \ldots, \tau_n)$ and $(x_1, x_2, \ldots, x_n)$.
$\boldsymbol{p}(t)$	vector $(p_1(t), p_2(t), \ldots, p_n(t))$.
$C_i(\tau_i, x_i)$ (or C_i)	cost of S_i for τ_i and x_i.
$c_i(\tau_i)$	cost of a component in S_i for τ_i.
α, β	desired level of $R_S(t)$; $0 < \alpha < 1$ and $\beta = \alpha^{1/n}$.
T	preassigned value of time.
C	specified maximum value of C_S.
ε	preassigned small value.
Δt	preassigned small value of time interval.
m	number of intervals into which the cost-range is divided.

$\boldsymbol{\tau}^*$, $\boldsymbol{X}^*$ optimal values of $\boldsymbol{\tau}$ and $\boldsymbol{X}$.

t^* value of t satisfying $R_S(t^*) = \alpha$ for the optimal sulution.

t^0 selected value of t; $t^0 \leq t$.

t_i (or $t_i(\tau_i, x_i)$) value of t satisfying $R_i(t_i) = \beta$ for τ_i and x_i.

$\boldsymbol{\tau}^0$, $\boldsymbol{X}^0$ solution for the problem of maximizing $\min_i\{t_i\}$.

k forward solution number, $k = 1, 2, 3, \ldots\ldots$

$\boldsymbol{\tau}^k$, $\boldsymbol{X}^k$ kth forward solution.

t^k value of t satisfying $R_S(t^k) = \alpha$ for the kth forward solution.

$R_S^k(t^{k-1})$ value of $R_S(t^{k-1})$ for the kth forward solution.

Σ_i, Π_i implies sum and product over all i ; $i = 1, 2, \ldots, n$.

The following assumptions are made after this section.

1. The system is partitioned into n statistically independent subsystems logically: the system functions if and only if all subsytems functions. Each subsystem S_i has x_i statistically independent and identically distributed components in parallel.

$$R_S(t) = \Pi_i R_i(t) = \Pi_i \{1 - (1 - p_i(t))^{x_i}\} \tag{6.1}$$

2. Each component has a constant mean-time-to-failure (or a constant failure rate), i.e.,

$$p_i(t) = \exp(-t/\tau_i) \qquad \text{for all } i. \tag{6.2}$$

3. Cost $c_i(\tau_i)$ is a continuous and monotonically increasing function of τ_i. C_S is linear in each x_i.

4. There is no repair.

6.4 Problem Formulation

First consider the problem of finding an optimal reliability design under cost constraint. Under the assumption of time-independent reliability, maximizing system reliability has incontestably been adopted as the performance criterion of the problem. For time-dependent reliability, what is the best performance criterion ? The following three criteria are candidates.

1) Maximizing mean time-to-system-failure (MTSF).

2) Maximizing system reliability $R_S(T)$ at time T.

3) Maximizing the time that $R_S(t)$ is above α, i.e., the value of t which satisfies $R_S(t) = \alpha$.

It is often required to lower the risk that systems with short time-to-system-failure (TSF) are produced. Criterion 1 is not always fit for this requirement, especially when the optimally designed system has a large variance of TSF. Moreover, the equation form for MTSF is complex, which makes the optimization intractable. In criterion 2, it matters how T is selected. Thus, this chapter adopts criterion 3.

The problem can be stated as follows.

Problem 1:

Find $\boldsymbol{\tau}^*$ and $\boldsymbol{X}^*$ to maximize the value of t which satisfies $R_S(t) = \alpha$ subject to

$$C_S = \Sigma_i C_i(\tau_i, x_i) = \Sigma_i x_i c_i(\tau_i) \leq C. \tag{6.3}$$

Similarly the optimization problem of minimizing system cost C_S is formulated. The constraint is that the value of t which satisfies $R_S(t) = \alpha$ is not less than a preassigned value T. Since $R_S(t)$ is monotonically decreasing with t, this constraint is equivalent to $R_S(T) \geq \alpha$. Accordingly the problem is stated as follows.

Problem 2:

Find $\boldsymbol{\tau}^*$ and $\boldsymbol{X}^*$ to minimize system cost C_S subject to

$$R_S(T) \geq \alpha. \tag{6.4}$$

6.5 Solution Method and Computational Procedure

First the solution method for Problem 1 is presented.

Let $\boldsymbol{\tau}^k$ and $\boldsymbol{X}^k$ be the optimal solution, which is named kth forward solution, for the problem of maximizing $R_S(t^{k-1})$ under the constraint (6.3), where t^k is the value of t satisfying $R_S(t^k) = \alpha$ for the kth forward solution, $k = 1, 2, 3, \ldots$. When t^0 is a value no more than t^*, the following relation holds.

$$t^0 \leq t^1 \leq t^2 \leq \ldots \leq t^* \tag{6.5}$$

The number sequence $\{t^k\}$ converges to t^*. As the forward solutions are obtained successively from the first one, we adopt, as the solution of Problem 1, the kth forward solution for which there exists the following relation for the first time.

$$R_S^k(t^{k-1}) - \alpha \leq \varepsilon \tag{6.6}$$

Thus, Problem 1 reduces to two problems of selecting t^0 and finding kth forward solution. If a better value is selected as t^0, we can reach the relation (6.6) more fast.

Selecting t^0

Assign the value $\beta = \alpha^{1/n}$ to each subsystem, and then from (6.1) and (6.2)

$$t_i(\tau_i, x_i) = -\tau_i \ln\{1 - (1 - \beta)^{1/x_i}\}. \tag{6.7}$$

Find the solution $\boldsymbol{\tau}^0$, $\boldsymbol{X}^0$ for the problem of maximizing $\min_i\{t_i\}$ under the constraint (6.3). Then we adopt, as t^0, the value of t which satisfies $R_S(t) = \alpha$ for the solution $\boldsymbol{\tau}^0$, $\boldsymbol{X}^0$.

The algorithm for finding $\boldsymbol{\tau}^0$ and $\boldsymbol{X}^0$ is as follows:

Step 1: Set $t_L = \max_{X \in \Omega} \{\min_i t_i(\tau_{i0}, x_i)\}$ and $t_U = \min_i t_i(\tau_{i0}, \nu)$, letting $\nu \equiv \left[\{C - \sum_{j=1(j\neq i)}^{n} c_j(\tau_{j0})\}/c_i(\tau_{i0})\right] + 1$, where $[\,y\,]$ is the maximum number no more than y, and Ω is the set of the vectors which satisfy the constraint (6.3) for τ_{i0}. Then go to Step 2.

Step 2: Set $t_a = (t_L + t_U)/2$ and obtain the values $(\tau_{i1}, 1)$, $(\tau_{i2}, 2)$, ..., (τ_{ir_i}, r_i) of (τ_i, x_i) which satisfy $t_i(\tau_i, x_i) = t_a$, letting $r_i = \left[\ \ln(1 - \beta) \,/\, \ln(1 - \exp[-t_a/\tau_{i0}])\right]$. If $r_i < 1$, then set $\tau_{ia} = \tau_{i0}$ and $x_{ia} = 1$. If $r_i \geq 1$, then set $\tau_{i,r_i+1} = \tau_{i0}$ and obtain

$$c_i(\tau_{ia})\, x_{ia} = \min_{x_i \in (1,2,\ldots,r_i+1)} \{c_i(\tau_{ix_i})\, x_i\}.$$

If $\Sigma_i\, c_i(\tau_{ia})\, x_{ia} \leq C$, then set $\tau_i^0 = \tau_{ia}$, $x_i^0 = x_{ia}$, $t_L = t_a$, and go to Step 2. If $\Sigma_i\, c_i(\tau_{ia})\, x_{ia} > C$, then set $t_U = t_a$, and go to Step 3.

Step 3: If $t_U - t_L > \Delta t$, then go to Step 2. If $t_U - t_L \leq \Delta t$, then terminate; the τ_i^0 and x_i^0 are the solution to be required.

In Step 1, t_L is obtained by setting 1 as the initial value of x_i and carrying on the operation of adding one redundant component to the subsystem with the minimum value of $t_i(\tau_{i0}, x_i)$ until the constraint (6.3) is violated.

Finding k-th forward solution

Since τ_i has an one-to-one correspondence to $p_i(t^{k-1})$, the problem of maximizing $R_S(t^{k-1})$ with respect to $\boldsymbol{\tau}$ and $\boldsymbol{X}$ is equivalent to the problem of maximizing $R_S(t^{k-1})$ with respect to $\boldsymbol{p}(t^{k-1})$ and $\boldsymbol{X}$, which is identical with time-independent optimal reliability design problem. This is a mixed-integer programming problem. Misra and Ljubojević [38] have presented a solution method for this problem by the Lagrange multiplier approach, which requires a differential cost function and often yields a solution having an excessive or lacking system cost since the value of x_i is obtained as a real number and then it is rounded off to the nearest integer number; therefore, it would be not appropriate to apply this solution method for finding kth forward solution. This chapter uses a solution method using dynamic programming, which yields a solution having the system cost equal to C and being optimal for a sufficiently large value of m.

Suppose that the allowable values of C_i, $i = 1, 2, \ldots, n$ and C are restricted to $l\Delta C$, $l = 1, 2, \ldots, m$, letting $\Delta C \equiv C/m$. Let τ_{il} and x_{il} be the values of τ_i and x_i which maximize $R_i(t^{k-1})$ under constraint $C_i \leq l\Delta C$. If there exist no values of τ_i and x_i which satisfy $C_i \leq l\Delta C$, then set τ_{i0} and 0 as the values of τ_{il} and x_{il}, respectively. Then the maximum-return functions $f_i(l\Delta C)$, $i = 1, 2, \ldots, n$ having the following recurrence relations are given.

$$f_1(l\Delta C) = 1 - (1 - \exp[-t^{k-1}/\tau_{1l}])^{x_{1l}} \quad \text{for } l = 1, 2, \ldots, m$$

$$f_i(l\Delta C) = \max_{j \in \{1,2,\ldots,l-i+1\}} [\{1 - (1 - \exp[-t^{k-1}/\tau_{ij}])^{x_{ij}}\} \times f_{i-1}(l\Delta C - j\Delta C)]$$

$$\text{for } i = 2, 3, \ldots, n;\ l = i, i+1, \ldots, m$$

Maximizing $R_i(t^{k-1})$ under $C_i \leq l\Delta C$ is equivalent to maximizing $R_i(t^{k-1})$ under $C_i = l\Delta C$, since both $R_i(t^{k-1})$ and C_i are monotonically increasing with τ_i and x_i. Accordingly τ_{il} and x_{il} can be obtained as follows:

Let $\bar{x}_{il}$ be the maximum one among the values of x_i which satisfy $c_i(\tau_{i0})x_i \leq l\Delta C$. Obtain the values of τ_i such that $c_i(\tau_i)x_i = l\Delta C$ for $x_i = 1, 2, \ldots, \bar{x}_{il}$, and select, as τ_{il} and x_{il}, the ones with the greatest value of $R_i(t^{k-1})$ among these $\bar{x}_{il}$ sets of values of τ_i and x_i.

Problem 2 is equivalent to the time-independent reliability design problem of minimizing system cost, since T is a specified value. Accordingly Problem 2 can be solved by the similar way to the solution method for finding the kth forward solution of Problem 1 mentioned above.

6.6 Illustrative Example

As an example of Problem 1, consider the system having four subsystems and the following cost functions ($\tau_{i0} = 2.0$ for $i = 1, 2, 3, 4$).

$$c_1(\tau_1) = 2.0 \exp(0.2\,\tau_1), \qquad c_2(\tau_2) = 3.0 \exp(0.3\,\tau_2),$$

$$c_3(\tau_3) = 2.5 \exp(0.1\,\tau_3), \qquad c_4(\tau_4) = 3.5 \exp(0.4\,\tau_4).$$

Let $\alpha = 0.95$, $C = 300$, $m = 50$ and $\Delta t = 0.0002$. The problem is to find $\boldsymbol{\tau} = (\tau_1, \tau_2, \tau_3, \tau_4)$ and $X = (x_1, x_2, x_3, x_4)$, to maximize the value of t which satisfies $R_S(t) = \alpha$ under the constraint $C_S = \Sigma_i\, c_i(\tau_i)\, x_i \leq 300$.

By the algorithm for selecting t^0, the t_L (or t_U) and C_{SL} (or C_{SU}) $= \Sigma_i\, c_i(\tau_{ia})\, x_{ia}$ are obtained successively as follows.

By Step 1,

$$t_L = 2.865, \qquad t_U = 4.443.$$

By the repetition of Step 2 and Step 3,

1. $t_L = 3.654$, $C_{SL} = 288.1$, 2. $t_U = 4.048$, $C_{SU} = 326.8$,
3. $t_U = 3.851$, $C_{SU} = 306.9$, 4. $t_L = 3.753$, $C_{SL} = 297.5$,
5. $t_U = 3.802$, $C_{SU} = 302.2$, 6. $t_L = 3.777$, $C_{SL} = 299.8$,
7. $t_U = 3.790$, $C_{SU} = 301.0$, 8. $t_U = 3.783$, $C_{SU} = 300.4$,
9. $t_U = 3.780$, $C_{SU} = 300.1$, 10. $t_L = 3.7787$, $C_{SL} = 299.94$,
11. $t_U = 3.7795$, $C_{SU} = 300.02$, 12. $t_U = 3.7791$, $C_{SU} = 299.98$,
12. $t_U = 3.7793$, $C_{SU} = 300.00$.

The $\boldsymbol{\tau}^0$, $\boldsymbol{X}^0$ and t^0 are obtained as follows.

$$\boldsymbol{\tau}^0 = (4.919,\ 3.950,\ 6.985,\ 3.385), \quad \boldsymbol{X}^0 = (\ 7,\ 9,\ 5,\ 11),$$

$$t^0 = 3.779.$$

By the algorithm for finding forward solution, the first forward solution is obtained as follows.

$$\boldsymbol{\tau}^1 = (4.904,\ 4.013,\ 8.755,\ 3.319), \quad \boldsymbol{X}^1 = (\ 9,\ 9,\ 5,\ 10),$$

$$t^1 = 3.924, \qquad R_S^1(t^0) = 0.95876.$$

The second forward solution is

$$\boldsymbol{\tau}^2 = (5.493,\ 4.013,\ 8.755,\ 3.582), \quad \boldsymbol{X}^2 = (\ 8,\ 9,\ 5,\ 9),$$

$$t^2 = 3.925, \qquad R_S^2(t^1) = 0.95012.$$

Of course, system cost is exactly 300 for every solution.

This chapter formulated only the reliability-maximization problem under cost-constraint and the cost-minimization problem under reliability-constraint, for a system with time-dependent reliability. But, it is possible to similarly formulate the other optimal reliability design problems, for example, the problem of having multiple constraints, the problem of having reliability functions other than exponential and the problem of having system cost that is nonlinear in x_i. The formulation would also be effective to the optimal design problem by stress-strength interference theory in which stress and strength are time-independent or time-dependent [64].

Conclusions

Developed in this thesis have been some methods for the reliability analysis and design of complex systems with the particular emphasis on fault tree and network techniques. The advantages of methods presented in this thesis are here summarized and additional remarks involving the problems left for future studies are made.

Chapter 1 extendedly defines the coherent structure function for logic trees containing mutually exclusive primary events and presents some properties useful for the qualitative and quantitative reliability analysis of complex systems. It should be emphasized that those properties are applicable to logic trees containing mutually dependent primary events(not only mutually exclusive), since they can be transformed into equivalent logic trees containing only mutually exclusive primary events. There are many cases where primary events are mutually dependent, e.g., the cases where 1) the occurrence of a component failure changes the stress level on the other components and therefore makes their failures easier to occur, 2) human errors are related to previous human errors or components failures, and 3) the interdependency among component failures does not hold due to design conditions, such as the use

of stand-by redundancy and the interdependency among repairs of failed components. It is also possible to discuss about modules, importances of components, shape of reliability (unreliability) function, etc., for logic trees containing mutually exclusive primary events, but such discussions are left for future studies.

System designers or system safety/reliability engineers often require a computer-aided means of organizing the knowledge about system safety/reliability so as to help them in making their decisions on system design or improvement as quickly and precisely as possible. In response to this requirement, Chapter 2 through Chapter 5 have presented some efficient methods for analyzing system reliability (unreliability) by using fault trees or networks. The list processing technique has been adopted throughout these chapters.

Various fault tree evaluations can be implemented through the combined use of reverse Polish sequence and tree sequence defined in Chapter 2. The use of those sequences considerably eases the constraints of storage requirements and computation time. Especially, invention of tree sequence has enabled us to analyze fault trees by means of a list processing that uses the recursive character of tree effectively. Furthermore, tree sequence is applicable not only to fault tree evaluation but also to the processing of all kinds of trees used in other applications, since the tree sequence completely represents the branching structure of tree.

Listing minimal cut sets is the elementary and important stage of fault tree evaluation, but it often takes too much computation time for large fault trees. The algorithm presented in Chapter 3 enumerates all minimal cut sets of fault trees more quickly than the former algorithms do. The computer program is ready. Needless to say, the list of minimal cut sets obtained can be used for computing the upper bound to the top event probability given in Chapter 1.

The method presented in Chapter 4 is a new approach to computing the exact value of the top event probability without using minimal cut sets. Reverse Polish sequence and tree sequence of the symbolic form of top event probability are obtained from the sequences of top event. Storing the symbolic form of top event probability is useful for repeated computations of probabilistic measures of fault tree. Typical examples of such applications are 1) the investigation of time-dependent behaviour of top event probability and some other probabilistic measures, 2) the estimation of confidence intervals of top event probability by the Monte Calro technique and 3) the computations of sensitivity coefficients of primary events and intermediate events. Application 2 is used for the fault tree evaluation with uncertain data. Application 3 is used for finding the optimal course of system improvement or fault diagnosis through computing the importance measures of events as shown in Sections 4.5, 4.6.

Left for future study is the development of an interactive fault tree analysis computer system using the methods presented in this thesis such that we could implement fault tree evaluations in real time. The gravest difficulty in the probabilistic evaluation of fault tree is the insufficiency of component failure data. How shall we evaluate fault trees and make decisions under the insufficiency of data ? How shall we collect failure data ? These are also the problems awaiting solutions.

Chapter 5 has shown an application of the list processing technique to the network reliability analysis. Reverse Polish sequence and tree sequence of the transmission Boolean function (TBF) of network are obtained by repeating the decomposition and reduction processes recursively. The probabilistic evaluation of network can be implemented by using the sequences of TBF instead of the sequences of top event in the method of Chapter 4. Though only node-pair reliability is treated in Chapter 5, other measures

concerning network reliability will be able to be anlyzed by the list processing technique if they can be expressed as a function of node and branch variables. The typical one of such measures is the probability that the network is connected, i.e., the probability that every node can communicate with every other node.

Chapter 6 has formulated an optimal design problem for a system with time-dependent reliability and has provided a solution method using dynamic programming. The mehtod is available to a system designer who has a hard time in making tradeoffs between system reliability and cost. Though only series systems have been considered in Chapter 6, the problem formulation can easily be extended to non-series systems. However, the solution method becomes complex and hence an approximate method will be required. For the future the optimal design problems of complex systems with repair and inspection need to be formulated and solved.

Appendix A

Proofs of Properties P1 - P4 in Section 1.3

Proof of P1:

Necessity: If $\chi(\boldsymbol{E}^{*(a)}) = 1$ for a fixed vector $\boldsymbol{E}^{*(a)}$, then there exists a term of $\chi(\boldsymbol{E}^*)$ composed of a nonempty subset of primary events having the value 1 in $\boldsymbol{E}^{*(a)}$. Therefore, $\chi(\boldsymbol{E}^*) = 1$ for all vectors $\boldsymbol{E}^*$ which satisfy $\boldsymbol{E}^* > \boldsymbol{E}^{*(a)}$. The relations (1.4) hold since any primary event belonging to $\bigcup_{i=1}^{n} \Omega_i^c$ does not appear in $\chi(\boldsymbol{E}^*)$.

Sufficiency: Assume there exists a term containing $E_{ij} \in \Omega_i^c$ in a s.o.p. form of $\psi(\boldsymbol{E})$. Let this term be QE_{ij} and the remaining terms be P in the lump, i.e., $T = P \vee QE_{ij}$. From (1.4), $T = P \vee Q(E_{i(\beta_i+1)} \vee E_{i(\beta_i+2)} \vee \cdots \vee E_{i(\alpha_i)})$ is also true. Thus $T = 1$ for a fixed vector $\boldsymbol{E}^{*(b)}$ in which every primary event appearing in Q has the value 1 and the remaining primary events are 0, regardless of the values of primary events belonging to $\bigcup_{i=1}^{n} \Omega_i^c$. Because $\psi(\boldsymbol{E})$ is monotone nondecreasing in $\boldsymbol{E}^*$, $T = 1$ for the vector obtained by changing the value of $E_{ij} \in \Omega_i$ from 0 to 1 in $\boldsymbol{E}^{*(b)}$. This fact means that $T = P \vee Q(E_{i1} \vee E_{i2} \vee \cdots \vee E_{i\alpha_i}) = P \vee Q$ is also true. Thus $\psi(\boldsymbol{E})$ can be represented as a s.o.p. form containing no primary events belonging to $\bigcup_{i=1}^{n} \Omega_i^c$. *Q.E.D.*

Proof of P2:

Let $\boldsymbol{E}^{*(c)}$ be a minimal cut vector. From the definition of minimal cut vector and P1, $\chi(\boldsymbol{E}^*) = 0$ for $\boldsymbol{E}^*$ which satisfies $\boldsymbol{E}^* < \boldsymbol{E}^{*(c)}$ and $\chi(\boldsymbol{E}^*) = 1$ for $\boldsymbol{E}^*$ which satisfies $\boldsymbol{E}^* > \boldsymbol{E}^{*(c)}$. Thus the logical product of primary events having the value 1 in $\boldsymbol{E}^{*(c)}$ is a term of $\chi(\boldsymbol{E}^*)$.

Let $C \equiv \{E_{i_1 j_1}, E_{i_2 j_2}, \ldots, E_{i_\eta j_\eta}\}$ be a set of primary events obtained as mentioned in P2. Because $E_{i_1 j_1} E_{i_2 j_2} \ldots E_{i_\eta j_\eta}$ is a term of $\chi(\boldsymbol{E}^*)$ and any term composed of a proper subset of C is not contained in $\chi(\boldsymbol{E}^*)$, then $\chi(\boldsymbol{E}^{*(c)}) = 1$ holds for $\boldsymbol{E}^{*(c)}$ such that all primary events belonging to C have the value 1 and the remaining primary events are 0, and $\chi(\boldsymbol{E}^*) = 0$ for $\boldsymbol{E}^*$ which satisfies $\boldsymbol{E}^* < \boldsymbol{E}^{*(c)}$. *Q.E.D.*

Proof of P3:

1) It is obvious from elementary set theory that (1.3) is equivalent to (A.1).

$$S(T = 1 | E_{ij} = 1) \subseteq S(T = 1 | E_{ik} = 1). \tag{A.1}$$

Eq. (A.1) is equivalent to (1.5).

2) The relation

$$\begin{aligned} &S(T = 0 | E_{ij} = 1) \cap S(T = 1 | E_{ik} = 1) = S(T = 1 | E_{ik} = 1) \\ &- S(T = 1 | E_{ij} = 1) \cap S(T = 1 | E_{ik} = 1) \end{aligned} \tag{A.2}$$

holds from elementary set theory. Eq. (1.6) is easily derived from (A.1) and (A.2).

3) $T = 1$ for the vector $\boldsymbol{E}^{*(c)}$ such that all primary events belonging to C have the value 1 and the remaining primary events are 0. From (1.5), we have

a) $T = 1$ from the vector $\boldsymbol{E}^{*(d)}$ such that, in $\boldsymbol{E}^{*(c)}$, the value of E_{ij} is changed into 0 and the value of E_{ij} into 1;

b) $T = 0$ for every vector $\boldsymbol{E}^*$ such that all primary events in a subset of $C - \{E_{ij}\}$ have the value 1 and the remaining primary events (including E_{ik}) are 0, since C is a minimal cut set.

Therefore, there exists a minimal cut vector $\boldsymbol{E}^{*(e)}$ which has $E_{ik} = 1$

and satisfies $E^{*(e)} \leq E^{*(d)}$. Thus, if $E_{ij} \xrightarrow{p} E_{ik}$, then P3-3 is true.

Suppose that P3-3 is true. Then, if $\psi(1_{ij}, E) = 1$ for a vector $(\bullet_i, E)$, then $\psi(1_{ik}, E) = 1$ holds for the same vector. This fact is equivalent to (1.5). Accordingly, $E_{ij} \xrightarrow{p} E_{ik}$ is true. *Q.E.D.*

Proof of P4:

Let G_{lm} be the logical product of primary events belonging to C_{lm} and let $G_l \equiv G_{l1} \vee G_{l2} \vee \cdots \vee G_{lt_l}$. From the addition rule of probability,

$$\Pr\{T = 1\} = \Pr\{G_1 \vee G_2 \vee \cdots \vee G_s = 1\} \leq \sum_{l=1}^{s} \Pr\{G_l = 1\}.$$

Since primary events contained in G_l are statistically independent, $G_{l1}, G_{l2}, \ldots, G_{lt_l}$ are associated [13, 6], i.e., $\mathrm{Cov}\{f, g\} \geq 0$ for all nondecreasing functions f and g of $G_{l1}, G_{l2}, \ldots, G_{lt_l}$. The following relation is derived from the former result [14] for associated binary variables.

$$\Pr\{G_l = 1\} \leq 1 - \prod_{m=1}^{t_l}(1 - \Pr\{G_{lm} = 1\}) = 1 - \prod_{m=1}^{t_l}(1 - P_{lm}).$$

Q.E.D.

Appendix B

Proofs of Properties P1 - P2 in Section 2.3

Proof of P.1:

Proof of 2): Let n_P be the number of primary events, n_S be the number of intermediate events (including the top event), and $-n_J$ be the sum of node elements. n_P is also equal to the sum of leaf elements, and n_S is equal to the number of logic gates. The number of the inputs of logic gates, or $n_J + n_S$, is equal to the number of primary events and intermediate events, or $n_P + n_S - 1$. Thus the sum of tree sequence TS(T) of top event T, or $n_P - n_J$, is equal to +1.

Proof of 1): Suppose that the sum becomes 0 or negative while adding. Let S be the subsequence of TS(T) from the first element to the element at which the sum becomes 0 or negative for the first time. Then the last element of S must be a node element, because, if not so, the sum from the first element to the just preceding element of the last element becomes negative. Let J_s be the last element of S, and Δ be the logic gate of P(T) corresponding to J_s. The elements of TS(T) corresponding to primary events and logic gates contained in the subtrees of the output of Δ are arranged before J_s (see Eq. (2.4)). Let SS be the subsequence of TS(T) that contains those elements and J_s. The sum of elements of SS is +1. Accordingly the sum of the subsequence obtained by removing SS from S is negative, and that subsequence contains the first element of TS(T). This fact is contradictory to the definition of S.

Proof of P.2:

Let $P^{(m+1)}$ be the subsequence obtained by removing the last element $-m$ from the tree sequence; the sum of $P^{(m+1)}$ is $m+1$.

It is obvious from Eq. (2.5) that there exists at least one way of partitioning $P^{(m+1)}$ into $m+1$ tree subsequences.

Let P_1, P_2, ..., P_{m+1} (rightward from the leftmost in order) be a set of tree subsequences obtained by partitioning $P^{(m+1)}$. Let $P^{(\eta)}$ be the subsequence from the first element of the tree sequence to the just preceding element of the first element of $P_{\eta+1}$, $\eta = 1, 2, ..., m$. The sum of $P^{(\eta)}$ is η and there exists the relation $P^{(\eta+1)} = P^{(\eta)} P_{\eta+1}$, where $P^{(1)} = P_1$. Accordingly, in order to show the uniqueness of the way of partitioning $P^{(m+1)}$ into $m+1$ tree subsequences, it is sufficient to show that $P^{(m+1)}$ is uniquely partitioned into $P^{(m)}$ and P_{m+1}. Suppose that there exists a way of partitioning $P^{(m+1)}$ different from the above, i.e., $P^{(m+1)} = \overline{P}^{(m)} \overline{P}_{m+1}$. If $\overline{P}_{m+1}$ (or P_{m+1}) is the subsequence of P_{m+1} (or $\overline{P}_{m+1}$), then the sum of the subsequence obtained by removing $\overline{P}_{m+1}$ (or P_{m+1}) from P_{m+1} (or $\overline{P}_{m+1}$) is 0, since both the sum of P_{m+1} and the sum of $\overline{P}_{m+1}$ are +1. This is contradictory to the fact that P_{m+1} (or $\overline{P}_{m+1}$) is a tree subsequence.

Q.E.D.

Appendix C

Proofs of 6 Checking Rules in Step 4 of ANCHEK Algorithm (Section 3.3)

Throughout the following proofs, c_{1i}, c_{2j} are used to represent elements belonging to C_1, C_2, respectively. Let $c' \equiv c_{1i} \cup c_{2j}$. From the definition in section 3.2, $c \equiv c_{1i*} \cup c_{2j*}$ is the set of primary events currently being checked.

Proof of Checking Rule 4.1:

Let $c \in F$. Then $c \in C_1 \otimes C_2$ since $c \in C_{1a}$ and $c \in C_{2a}$. If $c_{1i} \neq c$ (or $c_{2j} \neq c$) and $c' \subseteq c$, then $c_{1i} \subset c$ (or $c_{2j} \subset c$). This is contradictory to the fact that T_1 and T_2 are the reduced s.o.p. forms. Thus $c \in C$. It is apparent that the element belonging to $C_{1b} \cup C_{1c}$ or $C_{2b} \cup C_{2c}$ cannot be an element common in C_1 and C_2 since that element contains at least one non-common primary event.

Let $c \in F_{1a} \cup F_{1b}$ and $c \supset c_a \in C_{2a}$. Then $c \in C_1 \otimes C_2$ since $c \cup c_a = c$. If $c_{1i} \neq c$ and $c' \subset c$, then $c_{1i} \subset c$; this is contradictory. Suppose $c_{1i} = c \not\supset c_{2j}$. If $c' \subseteq c$, then $c_{2j} \subseteq c$; this is contradictory. Thus $c \in C$. The proof for the case of $c \in F_{2a} \cup F_{2b}$ is made similarly. An element $c_v \in C_{1c}$ (or C_{2c}) cannot be absorbed by any element belonging to C_{2a} (or C_{1a}) since c_v contains only non-common primary events. Also the element $c_w \in C_{1b} \cup C_{1c}$ (or $C_{2b} \cup C_{2c}$) cannot absorb any element belonging to C_2 (or C_1) since c_w contains at least one non-common primary event.

Proofs of Checking Rules 4.2 - 4.6:

Let $c \equiv c_{1i*} \cup c_{2j*}$ ($c_{1i*} \in C'_{1a} \cup C'_{1b} \cup C_{1c}$, $c_{2j*} \in C'_{2a} \cup C'_{2b} \cup C_{2c}$) and let $c_{1i} \neq c_{1i*}$, $c_{2j} \neq c_{2j*}$.

Consider the process of checking c by c_{1i}. Then the proofs of checking sub-rules a - d in Subroutine RULEA are given below.

a) Suppose the case of $c \supset c_{1i}$; $c = c_{1i}$ never holds, because $c_{1i*} \subseteq c_{1i}$ if $c = c_{1i}$. Then $c \supseteq c_{1i} \cup c_{2j*}$ since $c \supset c_{2j*}$. Thus $c \notin C$ if $c \neq c_{1i} \cup c_{2j*}$.

b, c) Suppose the case of $c \supset c_{1i}$ and $c = c_{1i} \cup c_{2j*}$. If $i < i*$, then it is not necessary to check c further since $c_{1i} \cup c_{2j*}$ is checked in its due order instead of c; thus $c \notin C$. If $i > i*$, then c_{1i} must be stored for additional checking.

d) Suppose the case of $c \not\supset c_{1i}$. If $c \supseteq c'$, then $c \supseteq c_{1i}$. But $c = c_{1i}$ never holds since it means $c_{1i*} \subseteq c_{1i}$. Thus $c \supset c_{1i}$. This is contradictory. Accordingly, if $c \not\supset c_{1i}$, then $c \not\supseteq c'$.

Similarly, for the process of checking c by c_{2j}, the proofs of checking sub-rules a - d are made.

Consider the case of $S_{1i*} \neq \phi$, $S_{2j*} \neq \phi$. Let $c_{1i} \in S_{1i*}$ and $c_{2j} \in S_{2j*}$. Then $c \supseteq c'$ since $c_{1i*} \cup c_{2j*} \cup c_{1i} \cup c_{2j} = c$. If $c \supset c'$, then $c \notin C$. If $c = c'$, then c must be checked further by other elements of $S_{1i*} \otimes S_{2j*}$ by applying the idempotence rule.

Next it is proved that, in each case of 4.2 - 4.6, it is not necessary to check c by all elements belonging to $C_1 \cup C_2$.

4.2) For $c_{1i} \in C_{1b} \cup C_{1c}$ (or $c_{2j} \in C_{2b} \cup C_{2c}$), $c \supset c_{1i}$ (or $c \supset c_{2j}$) never holds since c_{1i} (or c_{2j}) contains at least one non-common primary event.

4.3) Let $c_{1i*} \in C'_{1a}$ and $c_{2j*} \in C'_{2b}$. For $c_{1i} \in C_{1b} \cup C_{1c}$, $c \supset c_{1i}$ never holds since c_{1i} contains at least one non-common primary event contained in C_1. For $c_{2j} \in C_{2c}$, $c \supset c_{2j}$ never holds since c_{2j} contains at least one non-common primary event not contained in c_{2j*}.

The proof for the case of $c_{1i*} \in C'_{1b}$ and $c_{2j*} \in C'_{2a}$ is made similarly.

4.4) For $c_{1i} \in C_{1c}$ (or $c_{2j} \in C_{2c}$), $c \supset c_{1i}$ (or $c \supset c_{2j}$) never holds since c_{1i} (or c_{2j}) contains at least one non-common primary event not contained in c.

4.5) Let $c_{1i*} \in C_{1c}$ and $c_{2j*} \in C'_{2a} \cup C'_{2b}$. For $c_{1i} \in C_{1a}$, neither $c_{2j*} \subseteq c_{1i}$ nor $c_{2j*} \supseteq c_{1i}$ holds from the proof of 4.1. Thus c_{1i} contains at least one primary event not contained in c_{2j*}. For $c_{1i} \in C_{1c}$ (or $c_{2j} \in C_2$), c_{1i} (or c_{2j}) contains at least one primary event not contained in c, since neither $c_{1i} \subseteq c_{1i*}$ (or $c_{2j} \subseteq c_{2j*}$) nor $c_{1i} \supseteq c_{1i*}$ (or $c_{2j} \supseteq c_{2j*}$) holds and c_{1i} (or c_{2j}) contains no primary events contained in c_{2j*} (or c_{1i*}). For $c_{1i} \in C_{1b}$, $c_{1i} \cup c_{2j*} = c$ never holds since c_{1i*} contains at least one primary event not contained in both c_{1i} and c_{2j*}; therefore, checking sub-rules b & c are not necessitated.

The proof for the case of $c_{1i*} \in C'_{1a} \cup C'_{1b}$, $c_{2j*} \in C_{2c}$ is made similarly.

4.6) Let $c_{1i*} \in C_{1c}$ and $c_{2j*} \in C_{2c}$. For $c_{1i} \in C_1$ (or $c_{2j} \in C_2$), c_{1i} (or c_{2j}) contains at least one primary event not contained in c; therefore, $c \supseteq c_{1i}$ (or $c \supseteq c_{2j}$) never holds.

Q.E.D.

Appendix D

Derivation of Eqs. (4.14) and (4.15) in Section 4.3

From Table 2 in Chapter 2,

$$m = \frac{n^2 - 5n + 8}{2} + k + 1. \tag{D.1}$$

The following table gives m, $m' \equiv m - 3$, $n' \equiv n - 2$ for small values of n, k. .

n	3	4		5			6				7				
k	2	2	3	2	3	4	2	3	4	5	2	3	4	5	6
m	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
m'	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
n'	1	2		3			4				5				

There exist n' different members of m' for same n'. (We say that these members belong to n'-class.)

The last member of $(n'-1)$-class is $\frac{n'(n'-1)}{2}$, and that of n'-class is $\frac{n'(n'+1)}{2}$. Therefore, if m' belongs to n'-class, then

$$\frac{n'(n'-1)}{2} + 1 \leq m' \leq \frac{n'(n'+1)}{2}. \tag{D.2}$$

By solving this inequality with respect to n', we obtain

$$\frac{\sqrt{8m'+1}}{2} - \frac{1}{2} \leq n' \leq \frac{\sqrt{8(m'-1)+1}}{2} + \frac{1}{2}. \tag{D.3}$$

There exists only one integer n' satisfying inequality (D.3).

$$n' = \left[\frac{\sqrt{8(m' - 1) + 1}}{2} + \frac{1}{2} \right] \qquad \text{(D.4)}$$

Since $m' = m - 3$, $n' = n - 2$,

$$n = 2 + \left[\frac{\sqrt{8(m - 4) + 1}}{2} + \frac{1}{2} \right] \qquad \text{(D.5)}$$

From (D.1) and n computed by (D.5), we obtain k.

$$k = m - \frac{n^2 - 5n + 8}{2} - 1 \qquad \text{(D.6)}$$

References

[1] Aggarwal, K. K., "Redundancy Optimization in General System," *IEEE Trans. on Reliability*, Vol. R-25, pp.330-332, 1976.

[2] Allan, R. N., R. Billinton and M. F. De Oliveira, "An Efficient Algorithm for Deducing the Minimal Cuts and Reliability Indices of a General Network Configuration," *IEEE Trans. on Reliability*, Vol. R-25, pp.226-233, 1976.

[3] Barlow, R. E. and F. Proschan, *Mathematical Theory of Reliability*, N. Y.: Wiley, 1965.

[4] Barlow, R. E., J. B. Fussell and N. D. Singpurwalla (eds.), *Reliability and Fault Tree Analysis*, SIAM, 1975.

[5] Barlow, R. E. and H. E. Lambert, "Introduction to Fault Tree Analysis," *Reliability and Fault Tree Analysis*, SIAM, pp. 7-35, 1975.

[6] Barlow, R. E. and F. Proschan, *Statistical Theory of Reliability and Life Testing*, N. Y.: Holt, Rinehalt and Winston, 1975.

[7] Barlow, R. E. and F. Proschan, "Importance of System Components and Fault Tree Events," *Stochastic Process and Their Applications*, Vol. 3, pp.153-173, 1975.

[8] Bennetts, R. G., "On the Analysis of Fault Trees," *IEEE. Trans. on Reliability*, Vol. R-24, pp.175-185, 1975.

[9] Birnbaum, Z. W., J. D. Esary and S. C. Saunders, " Multi-Component Systems and Structures and Their Reliability," *Technometrics*, Vol. 3, pp.55-77, 1961.

[10] Brown, D. B., "A Computerized Algorithm for Determining the Reliability of Redundant Configurations," *IEEE Trans. on Reliability*, Vol. R-20, pp.121-124, 1971.

[11] Burdick, G. R., D. M. Rasmuson and S. L. Derby, "A Risk-Based Approach to Advanced Reactor Design," *IEEE Trans. on Reliability*, Vol. R-26, pp.198-202, 1977.

[12] Crosetti, P. A., "Fault Tree Analysis with Probability Evaluation," *IEEE Trans. on Nuclear Science*, Vol. NS-17, pp.465-471, 1970.

[13] Esary, J. D., F. Proschan and D. W. Walkup, "Association of Ramdom Variables with Applications," *Ann. Math. Statist.*, Vol. 38, pp.1466-1474, 1967.

[14] Esary, J. D. and F. Proschan, "Coherent Structures of Non-identical Components," *Technometrics*, Vol. 5, pp.191-209, 1963.

[15] Frank, H. and I. T. Frisch, *Communication, Transmission, and Transportation Networks*, Addison-Wesley, 1971.

[16] Fratta, L. and U. G. Montanari, "A Boolean Algebra Method for Computing the Terminal Reliability in a Communication Network," *IEEE Trans. on Circuit Theory*, Vol. CT-20, pp.203-211, 1973.

[17] Fussell, J. B. and W. E. Vesely, "A New Methodology for Obtaining Cut Sets for Fault Trees," *Trans. American Nuclear Society*, Vol. 15, pp.262-263, 1972.

[18] Fussell, J. B., "A Formal Methodology for Fault Tree Construction," *Nuclear Sci. and Eng.*, Vol. 52, pp.421-432, 1973.

[19] Fussell, J. B., G. J. Powers and R. G. Bennetts, "Fault Trees - A State of the Art Discussion," *IEEE Trans. on Reliability*, Vol. R-23, pp.51-55, 1974.

[20] Fussell, J. B., "How to Hand-Calculate System Reliability and Safety Characteristics," *IEEE Trans. on Reliability*, Vol. R-24, pp.169-174, 1975.

[21] Fussell, J. B., "Fault Tree Analysis - Concepts and Technique," *Generic Techniques in System Reliability Assessment*, Nooddhoff, pp.133-162, 1976.

[22] Fussell, J. B. and G. R. Burdick (eds.), *Nuclear Systems Reliability Engineering and Risk Assessment*, SIAM, 1977.

[23] Haasl, D. F., "Advanced Concepts in Fault Tree Analysis," *System Safety Symposium*, Seattle, Wash.: The Boeing Company, 1965.

[24] Hänsler, E., "A Fast Recursive Algorithm to Calculate the Reliability of a Communication Network," *IEEE Trans. on Communication*, Vol. COM-20, pp.637-640, 1972.

[25] Henley, E. J. and R. A. Williams, *Graph Theory in Modern Engineering*, Academic Press, 1973.

[26] Henley, E. J. and J. Lynn (eds.), *Generic Techniques in Systems Reliability Assessment*, Noordhoff, 1974.

[27] Gangloff, W. C., " 'What Ifs ?' for Nuclear Plants," *IEEE Spectrum*, Vol. 14, No.6, pp.53-58, 1977.

[28] Garribba, S., P. Mussio, F. Naldi, G. Reina and G. Volta, "Efficient Construction of Minimal Cut Sets from Fault Trees," *IEEE Trans. on Reliability*, Vol. R-26, pp.88-94, 1977.

[29] Inagaki, T., K. Inoue and H. Akashi, "Interactive Optimization of System Reliability under Multiple Objectives,"

IEEE Trans. on Reliability, Vol. R-27, pp.264-267, 1978.

[30] Jensen, P. A. and M. Bellmore, "An Algorithm to Determine the Reliability of a Complex System," *IEEE Trans. on Reliability*, Vol. R-18, pp.169-174, 1969.

[31] Knuth, D. H., *The Art of Computer Programming*, Reading, Mass.: Addison-Wesley, 1968.

[32] Koen, B. V. and A. Carnino, "Reliability Calculation with a List Processing Technique," *IEEE Trans. on Reliability*, Vol. R-23, pp.43-50, 1974.

[33] Krishnamurthy, E. V. and G. Komissar, "Computer-Aided Reliability Analysis of Complicated Networks," *IEEE Trans. on Reliability*, Vol. R-21, pp.86-89, 1972.

[34] Lambert, H. E., "Fault Trees for Decision Making in System Analysis," Lawrence Livermore Lavoratory, Report UCRL-51829, 1975.

[35] Lapp, S. A. and G. J. Powers, "Computer-Aided Synthesis of Fault-Trees," *IEEE Trans. on Reliability*, Vol. R-26, pp.2-13, 1977.

[36] Mine, H., "Reliability of Physical System," *Trans. of 1959 Internat. Symp. on Circuit and Information Theory*, IRE, pp.138-151, 1959.

[37] Misra, K. B., "An Algorithm for the Reliability Evaluation of Redundant Networks," *IEEE Trans. on Reliability*, Vol. R-19, pp.146-151, 1970.

[38] Misra, K. B. and M. D. Ljubojevic, "Optimal Reliability Design of a System: a New Look," *IEEE Trans. on Reliability*, Vol. R-22, pp.255-258, 1973.

[39] Moore, E. F. and C. E. Shannon, "Reliable Circuits Using Less Reliable Relays," *J. Franklin Institute*, Vol. 262,

pp.191-208 (Part I), pp.281-297 (Part II), 1956.

[40] Moskowitz, F. and J. B. Mclean, "Some Reliability Aspects of System Design," *IRE Trans. on Reliability and Quality Control*, Vol. PGRQC-8, pp.7-35, 1956.

[41] Murchland, J. D., "A Moment Method for the Calculation of a Confidence Interval for the Failure Probability of a System," *Proc. 1972 Annual Reliability and Maintainability Symp.*, pp.565-577, 1972.

[42] Murchland, J. D., "Fundamental Concepts and Relations for Reliability Analysis of Multi-State Systems," *Reliability and Fault Tree Analysis*, SIAM, 1975.

[43] Nakagawa, Y. and K. Nakashima, "A Heuristic Method for Determining Optimal Reliability Allocation," *IEEE Trans. on Reliability*, Vol. R-26, pp.156-161, 1977.

[44] Nakagawa, Y., K. Nakashima and Y. Hattori, "Optimal Reliability Allocation by Branch-and-Bound Technique," *IEEE Trans. on Reliability*, Vol. R-27, pp.31-38, 1978.

[45] Nakagawa, Y. and Y. Hattori, "Optimal Design of High Reliable System: From the Point of View of Mathematical Programming," *Communications of the Operation Research Society of Japan*, Vol. 23, pp.536-543, Sep. 1978 (in Japanese).

[46] Nakagawa, Y. and Y. Hattori, "Reliability Optimization with Multiple Properties and Integer Variables," *IEEE Trans. on Reliability*, Vol. R-28, pp.73-78, 1979.

[47] Nakashima, K. and Y. Hattori, "System Reliability Analysis by Using Fault Tree," Reports of Professional Groups on Circuit and System Theory of IECE, CST75-49, pp.71-78, July 1975 (in Japanese).

[48] Nakashima, K. and Y. Hattori, " A Consideration on the Fault

Tree Analysis," Reports of Professional Group on Reliability of IECE, R76-19, pp.1-7, Oct. 1976 (in Japanese).

[49] Nakashima, K. and Y. Hattori, "Analysis of Fault Trees by Using Tree Sequences," *Trans. of IECE of Japan*, Vol. E60, pp.175-182, 1977.

[50] Nakashima, K. and K. Yamato, "Optimal Design of a Series-Parallel System with Time-Dependent Reliability," *IEEE Trans. on Reliability*, Vol. R-26, pp.119-120, 1977.

[51] Nakashima, K. and Y. Hattori, "An Efficient Algorithm for Analyzing Fault Tree," Reprints of 1978-Oct. Meeting of the Operation Research Society of Japan, B-2, pp.60-61, Oct. 1978 (in Japanese).

[52] Nakashima, K. and Y. Hattori, "A Method for Representing the Transmission Boolean Function of Network," *Proc. 1979 International Symposium on Circuits and Systems*, pp.456-457, July 1979.

[53] Nakashima, K., "Some Properties of Logic Trees Containing Mutually-Exclusive Primary-Events," *IEEE Trans. on Reliability*, Vol. R-28, pp.303-308, 1979.

[54] Nakashima, K. and Y. Hattori, "An Efficient Bottom-Up Algorithm for Enumerating Minimal Cut Sets of Fault Tree," *IEEE Trans. on Reliability*, Vol. R-28, pp.353-357, 1979.

[55] Nakashima, K. and Y. Hattori, "Supplement to 'An Efficient Bottom-Up Algorithm for Enumerating Minimal Cut Sets of Fault Tree' " NAPS Document NO. 03469, 28 pages, ASIS-NAPS; Microfiche Publication; PO Box 3513, Grand Central Station; New York, NY 10017 USA, 1979.

[56] Neogy, R., "Fault Trees in Ocean Systems," *Proc. 1975 Annual Reliability and Maintainability Symp.*, pp.280-285, 1975.

[57] Nielsen, D., "Use of Cause-Consequence Chart in Practical Systems Analysis," *Reliability and Fault Tree Analysis*, SIAM, pp.849-880, 1975.

[58] Phibbs, E. and S. H. Kuwamoto, "An Efficient Map Method for Processing Multistate Logic Trees," *IEEE Trans. on Reliability*, Vol. R-21, pp.93-98, 1972.

[59] Powers, G. J. and F. W. Tompkins, "Fault Tree Synthesis for Chemical Processes," *AIChE Journal*, Vol. 20, pp.376-387, 1974.

[60] Premo, A. F., "The Use of Boolean Algebra and Truth Table in the Formulation of a Mathematical Model of Success," *IEEE Trans. on Reliability*, Vol. R-12, pp.45-49, 1963.

[61] Rasmuson, D. M. and N. H. Marshall, "FATRAM - A Core Efficient Cut-Set Algorithm," *IEEE Trans. on Reliability*, Vol. R-27, pp.250-253, 1978.

[62] Rosenthal, A., "Approaches to Comparing Cut-Set Enumeration Algorithms," *IEEE Trans. on Reliability*, Vol. R-28, pp.62-65, 1979.

[63] Sakawa, M., "Multiobjective Reliability and Redundancy Optimization of a Series-Parallel System by the Surrogate Worth Trade-Off Method," *Microelectronics and Reliability*, Vol. 17, pp.465-467, 1978.

[64] Taraman, S. I. and K. C. Kapur, "Optimization Considerations in Design Reliability by Stress-Strength Interference Theory," *IEEE Trans. on Reliability*, Vol. R-24, pp.136-138, 1975.

[65] Tillman, F. A., C. L. Hwang and W. Kuo, "Optimization Techniques for System Reliability with Redundancy - A Review," *IEEE Trans. on Reliability*, Vol. R-26, pp.148-155, 1977.

[66] Tillman, F. A., C. L. Hwang and W. Kuo, "Determining Component Reliability and Redundancy for Optimum System Reliability," *IEEE Trans. on Reliability*, Vol. R-26, pp.162-165, 1977.

[67] Tsuboi, T. and K. Aihara, "A New Approach to Computing the Terminal Reliability in Large Complex Networks," *Trans. of IECE of Japan*, Vol. 58-A, pp.783-790, 1975 (in Japanese).

[68] U.S. Atomic Energy Commission, "Reactor Safety Study - An Assessment of Accident Risks in U.S. Commercial Nuclear Power Plants," WASH-1400, 1974.

[69] Vesely, W. E., "A Time-Dependent Methodology for Fault Tree Evaluation," *Nuclear Eng. and Design*, Vol. 13, pp.337-360, 1970.

[70] Vesely, W. E. and R. E. Narum, "PREP and KITT: Computer Codes for the Automatic Evaluation of a Fault Tree," IN-1349, Idaho Nuclear Corporation; Idaho Falls; Idaho USA, 1970.

[71] Wheeler, D. B., J. S. Hsuan, R. R. Duersch and G. M. Roe, "Fault Tree Analysis Using Bit Manipulation," *IEEE Trans. on Reliability*, Vol. R-26, pp.95-99, 1977.

[72] Worrell, R. B., "Using the Set Equation Transformation System in Fault Tree Analysis," *Reliability and Fault Tree Analysis*, SIAM, pp.165-185, 1975.